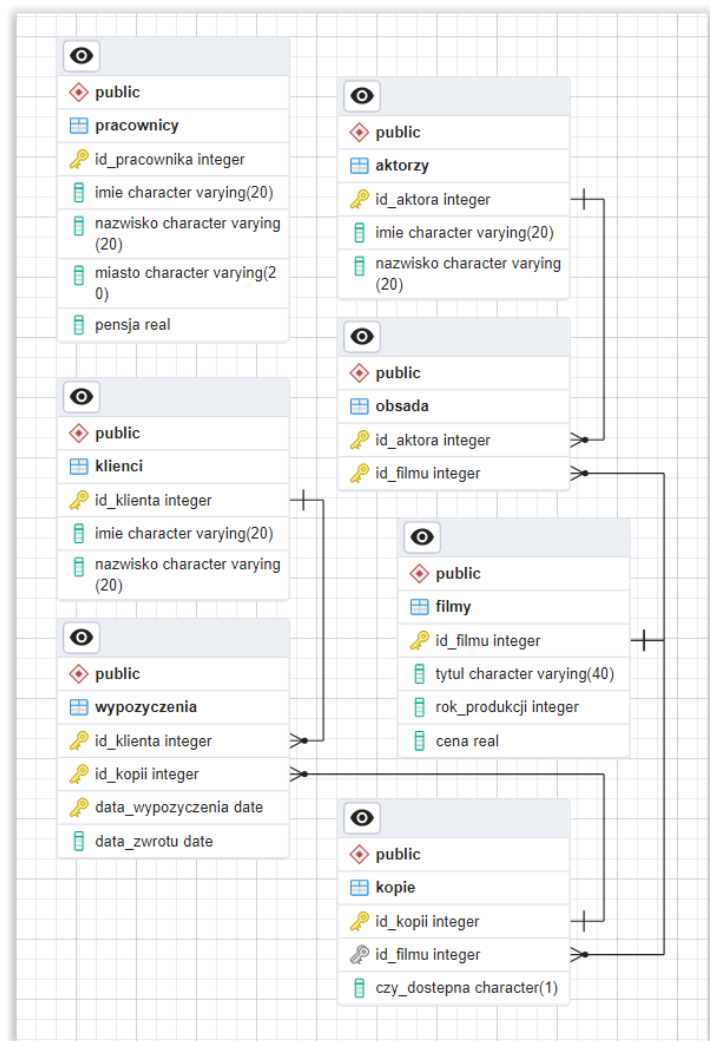


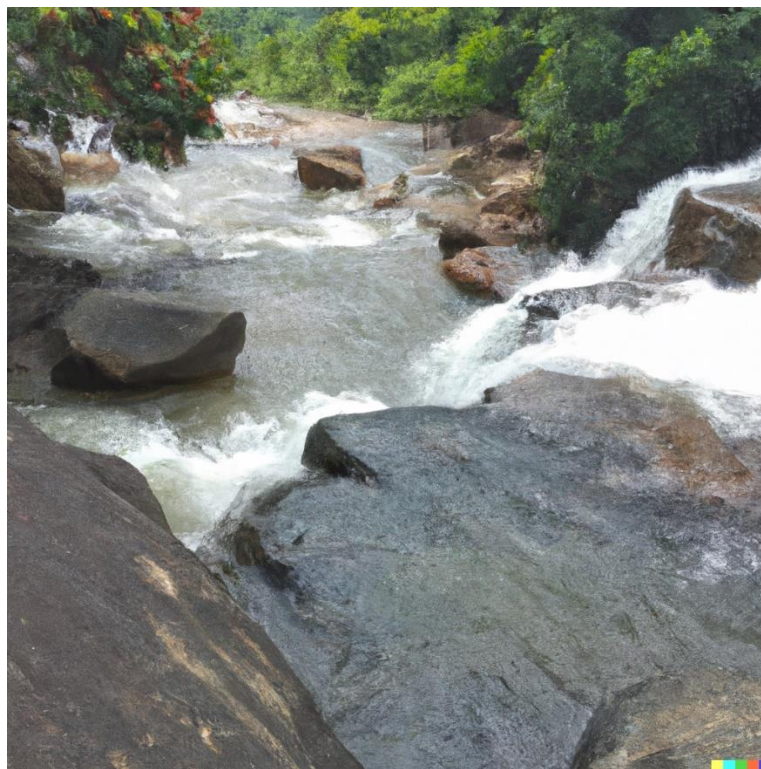
Przetwarzanie strumieni danych w systemach Big Data wprowadzenie

Krzysztof Jankiewicz

Jaki jest świat danych?



Wsadowy czy strumieniowy?



Dlaczego?

Już w 2015 roku Databricks przeprowadziła badanie wśród swoich użytkowników dotycząca wykorzystywania przez nich mechanizmów Spark Streaming

Okazało się, że 56% z nich korzysta z nich, a 48% uważa, że jest kluczowy element w ich biznesie.

Dlaczego?

czyli jak slajd nie
powinien wyglądać

- ***Real-time analytics***: Przetwarzanie strumieni danych umożliwia analizę danych w czasie rzeczywistym, co umożliwia przedsiębiorstwom natychmiastowe reagowanie na zmiany w danych i podejmowanie lepszych decyzji biznesowych. Dzięki temu firmy mogą szybciej dostosowywać swoje strategie i podejmować odpowiednie działania, co przekłada się na poprawę wyników finansowych.
- ***Efektywność operacyjna***: Przetwarzanie strumieni danych pozwala na automatyzację procesów biznesowych i eliminację opóźnień w przetwarzaniu danych, co przekłada się na znaczne usprawnienie działalności przedsiębiorstw. Dzięki temu zespoły mają więcej czasu na podejmowanie decyzji biznesowych i rozwijanie strategii.
- ***Zwiększenie konkurencyjności***: Przetwarzanie strumieni danych umożliwia przedsiębiorstwom szybkie i efektywne wykorzystanie danych do podejmowania decyzji biznesowych. Dzięki temu firmy mogą lepiej dostosować się do zmieniającego się otoczenia rynkowego i zwiększyć swoją konkurencyjność.
- ***Ulepszona jakość usług***: Przetwarzanie strumieni danych umożliwia przedsiębiorstwom monitorowanie jakości swoich usług w czasie rzeczywistym i natychmiastowe reagowanie na sytuacje awaryjne. Dzięki temu przedsiębiorstwa mogą zwiększyć zadowolenie klientów poprzez zapewnienie im lepszej jakości usług.
- ***Redukcja kosztów***: Przetwarzanie strumieniowe strumieni danych przedsiębiorstwom na automatyzację procesów biznesowych i eliminację opóźnień w przetwarzaniu danych, co przekłada się na redukcję kosztów operacyjnych. Dodatkowo, dzięki szybszemu dostępowi do informacji, firmy mogą podejmować decyzje biznesowe na podstawie bardziej aktualnych danych, co pozwala uniknąć kosztownych błędów.
- ***Możliwość szybkiego reagowania na problemy***: Przetwarzanie strumieni danych pozwala na szybkie wykrycie problemów w procesach biznesowych i natychmiastowe podjęcie działań mających na celu ich rozwiązanie. Dzięki temu przedsiębiorstwa mogą uniknąć kosztownych przerw w działalności i utrzymanie ciągłości biznesowej.

Przetwarzanie Strumieni Danych

- What,
 - Where,
 - When, and
 - How
- of Large-Scale Data Processing

Streaming Systems

Tyler Akidau, Slava Chernyak, Reuven Lax
O'Reilly Media, II wydanie, 2019

Wstęp

- Czym jest strumień?
- Czym jest silnik/system przetwarzania strumieni danych?
- Aplikacje przetwarzające strumienie danych

Czym jest strumień?

- Dwa wymiary pojęcia: liczność i natura
- Liczność
 - Dane ograniczone (*bounded data*) – zbiór danych, który ma skończony rozmiar
 - **Dane nieograniczone** (*unbounded data*) – zbiór danych, który ma nieskończony (teoretycznie) rozmiar
- Natura (charakter)
 - Tabela – zbiór danych z określonego momentu w czasie
 - **Strumień** – dane, które prezentują, element po elemencie, ewolucję (zmianę) danych w czasie.

Przykłady strumieniowych źródeł danych

- Transakcje giełdowe w czasie rzeczywistym
- Zarządzanie zapasami w handlu
- Aplikacje do udostępniania przejazdów
- Gry dla wielu graczy
- Internet of Things
- Systemy śledzenia lokalizacji
- Transakcje bankowe

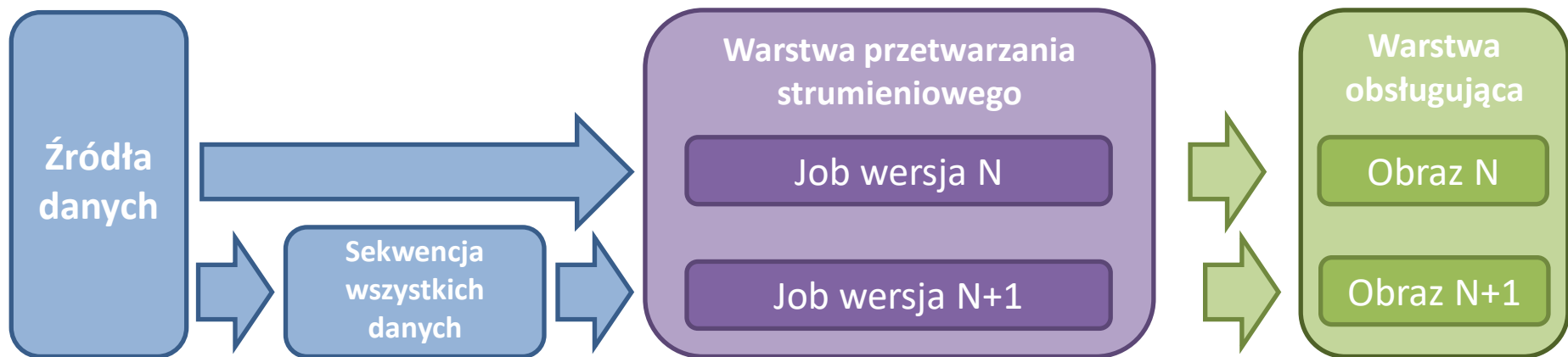
Czym jest silnik/system przetwarzania strumieni danych?

Typ silnika przetwarzania danych zaprojektowany z myślą o nieskończonych zbiorach danych

Przetwarzanie *Batch*, *Micro-Batch*, *Stream*

- Skończone zbiory danych – przetwarzanie wsadowe.
- nieskończone zbiory danych
 - Przetwarzanie mikro-wsadowe
 - stałe okna – problem kompletności danych
 - okna sesyjne – rozbieżność danych pomiędzy mikro-wsadami
 - Przetwarzanie strumieni danych (ciągłe/pełnowymiarowe)

Architektury systemów Big Data – przypomnienie



Ewolucja systemów przetwarzania strumieni danych

Początki w ramach projektów badawczych, ale także komercyjnych 1990

Generacje rozproszonych systemów przetwarzania strumieni danych:

- Pierwsza generacja (2011)
 - małe opóźnienia
 - niskopoziomowe API
 - **brak obsługi etykiet zdarzeń** – brak powtarzalności, spójności i dokładności wyników
 - gwarancje "at-least once"
 - wykorzystanie w architekturze Lambda

- Druga generacja (2013)
 - API wysokiego poziomu
 - lepsza obsługa awarii
 - zwiększona przepustowość
 - zwiększone opóźnienia
 - nadal oparcie się na czasie i kolejności przybywania zdarzeń
- Trzecia generacja (2015)
 - **wykorzystanie etykiet zdarzeń**
 - gwarancje "exactly-once"
 - możliwość konfigurowania przepustowości/opóźnienia
 - możliwość obsługi danych bieżących oraz historycznych
 - wyniki powtarzalne, spójne i dokładne
 - możliwe wykorzystanie architektury Kappa

Czym się charakteryzują systemy przetwarzania strumieni danych (*Stream Data Processing*)?

- Odbierają dane w sposób ciągły
- Działają 24/7 – duża waga mechanizmów obsługi awarii
- Przetwarzają dane na bieżąco
- Wyniki dostępne są z tzw. niską latencją
- Miejsca docelowe o dodatkowej funkcjonalności
 - Duża częstotliwość operacji
 - Możliwa potrzeba aktualizacji danych (a nie tylko dopisywania nowych)
- Możliwe ograniczenia dotyczące dokładności wyniku, stosowania przybliżeń, heurystyk
- Stosunkowo mniejsza przepustowość (rozłożona w czasie)

Aplikacje przetwarzające strumienie danych

Przypadki zastosowań

- Analiza lokalizacji
- Wykrywanie oszustw
- Transakcje giełdowe w czasie rzeczywistym
- Marketing, sprzedaż i analityka biznesowa
- Aktywność klienta / użytkownika (aplikacji, portalu, urządzeń)
- Monitorowanie i raportowanie wewnętrznych systemów informatycznych
- Monitorowanie dzienników: rozwiązywanie problemów z systemami, serwerami, urządzeniami i nie tylko
- Uczenie maszynowe i sztuczna inteligencja: łączenie przeszłych i obecnych danych
- SIEM (Security Information and Event Management): analizowanie dzienników i danych o zdarzeniach w czasie rzeczywistym w celu monitorowania, pomiarów i wykrywania zagrożeń
- Zapasy w handlu detalicznym / magazynie: zarządzanie zapasami we wszystkich kanałach i lokalizacjach oraz zapewnianie bezproblemowej obsługi na wszystkich urządzeniach
- Zarządzanie pojazdami współdzielonymi: łączenie danych dotyczących lokalizacji, użytkownika i cen na potrzeby analiz predykcyjnych; dopasowywanie kierowców do najlepszych kierowców pod względem bliskości, miejsca docelowego, cen i czasu oczekiwania

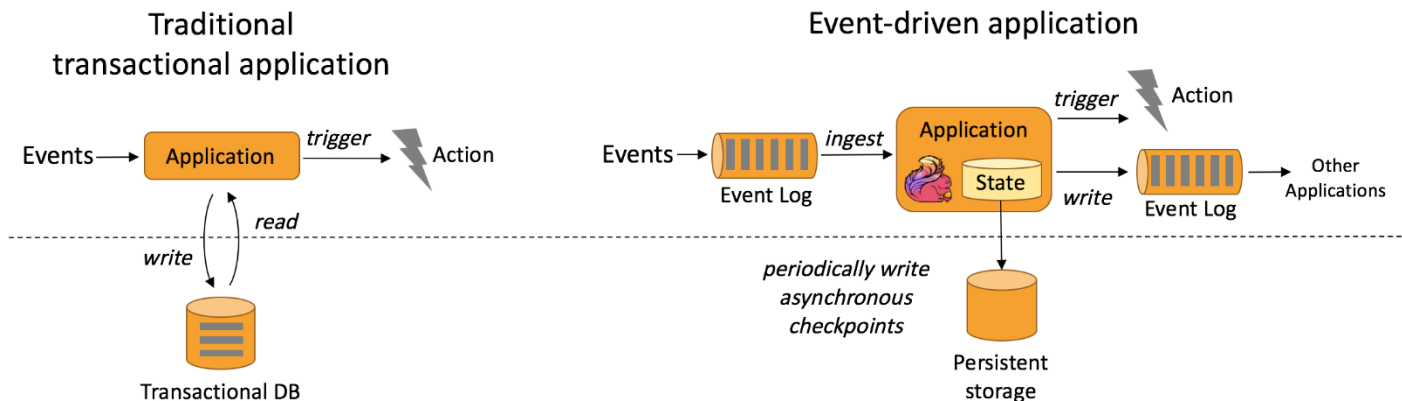
Główne przykłady zastosowań

- Aplikacje oparte na zdarzeniach (rekomendacje czasu rzeczywistego, wykrywanie wzorców, CEP, wykrywanie anomalii)
- Przepływy danych
- Analityka strumieni danych (monitorowanie jakości, zachowania użytkowników)

Aplikacje oparte na zdarzeniach

Event-driven Applications

- Cechy
 - odczyt z jednego lub wielu źródeł strumienia zdarzeń
 - natychmiastowa reakcja na pojawiające się zdarzenia
- Rozwiązania klasyczne vs strumieniowe



- Zalety
 - znacznie większa przepustowość
 - mniejsze opóźnienia

- Przykłady:
 - detekcja anomalii
 - detekcja oszustw
 - alarmy, których definicje oparte są na regułach
 - monitorowanie przetwarzania procesów biznesowych
 - aplikacje internetowe (analiza ruchu, aktywności użytkowników)

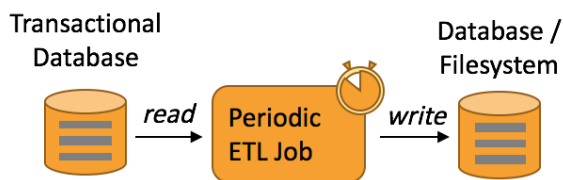
<https://flink.apache.org/usecases.html>

Przepływy danych

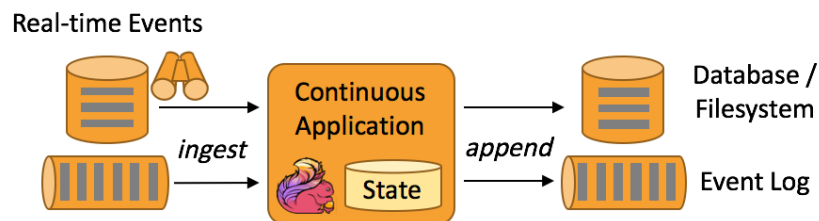
Data Pipeline Applications

- Cechy
 - odpowiednik operacji ETL
 - transformacja i przenoszenie danych realizowane w sposób ciągły
- Rozwiązania klasyczne vs strumieniowe

Periodic ETL



Data Pipeline



<https://flink.apache.org/usecases.html>

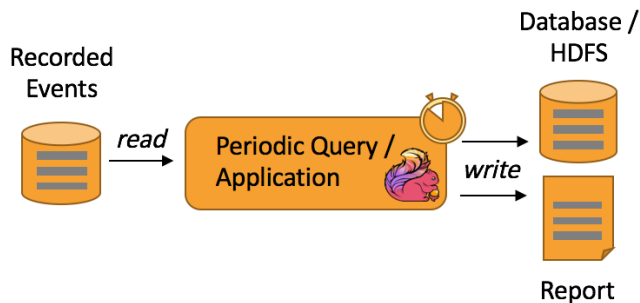
- Zalety
 - mniejsze opóźnienia
 - znacznie większa liczba przypadków użycia
- Główne obszary zastosowań
 - tworzenie indeksów dla silników wyszukiwani
 - ETL czasu rzeczywistego

Analityka strumieni danych

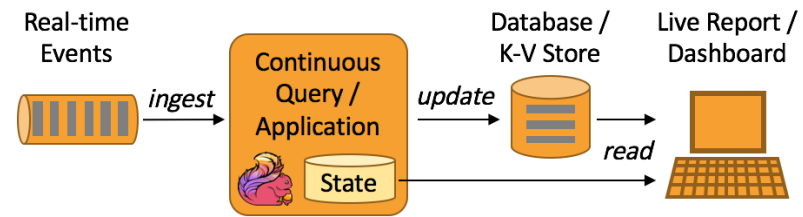
Data Analytics Applications

- Cechy
 - analiza danych pochodzących ze źródeł
 - wyniki analiz są zapisywane lub bezpośrednio prezentowane
- Rozwiązania klasyczne vs strumieniowe

Batch analytics



Streaming analytics



<https://flink.apache.org/usecases.html>

- Zalety:
 - zmniejszone opóźnienia
 - prostsza architektura
- Przykłady zastosowań
 - Monitorowanie jakości usług (np. sieci telekomunikacyjnych)
 - Analiza danych grafowych dużej skali
 - Analiza danych szybko zmieniających się (np.: zachowania użytkowników, współpracowników, maszyn)

Pytania?

Przetwarzanie strumieni danych w systemach Big Data

część 2 – podstawy

Krzysztof Jankiewicz

Plan

- Architektury i generacje silników – przypomnienie
- Poziomy abstrakcji – wprowadzenie
- Budowa aplikacji
- Źródła i ich typy
- Transformacje i ich typy
- Przetwarzanie stanowe – wprowadzenie
- Ujścia
- Gwarancje
- Czas – kilka pytań

Architektury silników

- Micro-batch
- Pełnostrumieniowe
- *Job-per-cluster* – pojedyncze zadanie alokuje zasoby (tworzy klaster oparty na JVMs) i dokonuje przetwarzania strumieni w odseparowanym środowisku
- *Session-per-cluster* – klaster alokuje zasoby, a następnie obsługuje zadania wchodzące w ramach pojedynczej sesji użytkownika. Zasoby są dzielone pomiędzy zadania jednej sesji
- *Server-per-cluster* – klaster alokuje zasoby, a następnie obsługuje zadania pochodzące z wielu sesji (np. wielu użytkowników). Zasoby są dzielone pomiędzy zadania wielu sesji

Ewolucja systemów przetwarzania strumieni danych

Początki w ramach projektów badawczych, ale także komercyjnych 1990

Generacje rozproszonych systemów przetwarzania strumieni danych:

- Pierwsza generacja (2011)
 - małe opóźnienia
 - **niskopoziomowe API**
 - **brak obsługi etykiet zdarzeń** – brak powtarzalności, spójności i dokładności wyników
 - gwarancje "at-least once"
 - wykorzystanie w architekturze Lambda

- Druga generacja (2013)
 - **API wysokiego poziomu**
 - lepsza obsługa awarii
 - zwiększona przepustowość
 - zwiększone opóźnienia
 - nadal oparcie się na czasie i kolejności przybywania zdarzeń
- Trzecia generacja (2015)
 - **wykorzystanie etykiet zdarzeń**
 - gwarancje "exactly-once"
 - możliwość konfigurowania przepustowości/opóźnienia
 - możliwość obsługi danych bieżących oraz historycznych
 - wyniki powtarzalne, spójne i dokładne
 - możliwe wykorzystanie architektury Kappa

Poziomy abstrakcji – wprowadzenie

- Niskopoziomowe API – "ręczna" obsługa zdarzeń, "ręczne" utrzymywanie stanu przetwarzania
- Podstawowe API – *DataStream API*
 - nieskończone kolekcje napływających obiektów
 - przetwarzanie funkcyjne oparte o przetwarzanie kolekcji
 - duże możliwości
- Wysokopoziomowe API
 - *Table API*
 - model relacyjny
 - źródło to "tabela", w której pojawiają z upływem czasu nowe dane
 - metody odpowiadające klauzulom SQL
 - *SQL*
 - model relacyjny
 - źródło jak wyżej
 - wykorzystanie składni SQL
- *Complex Event Processing* – wykorzystanie wzorców (np. MR)

Użyte nazwy API są umowne i w różnych systemach mogą nazywać się odmiennie

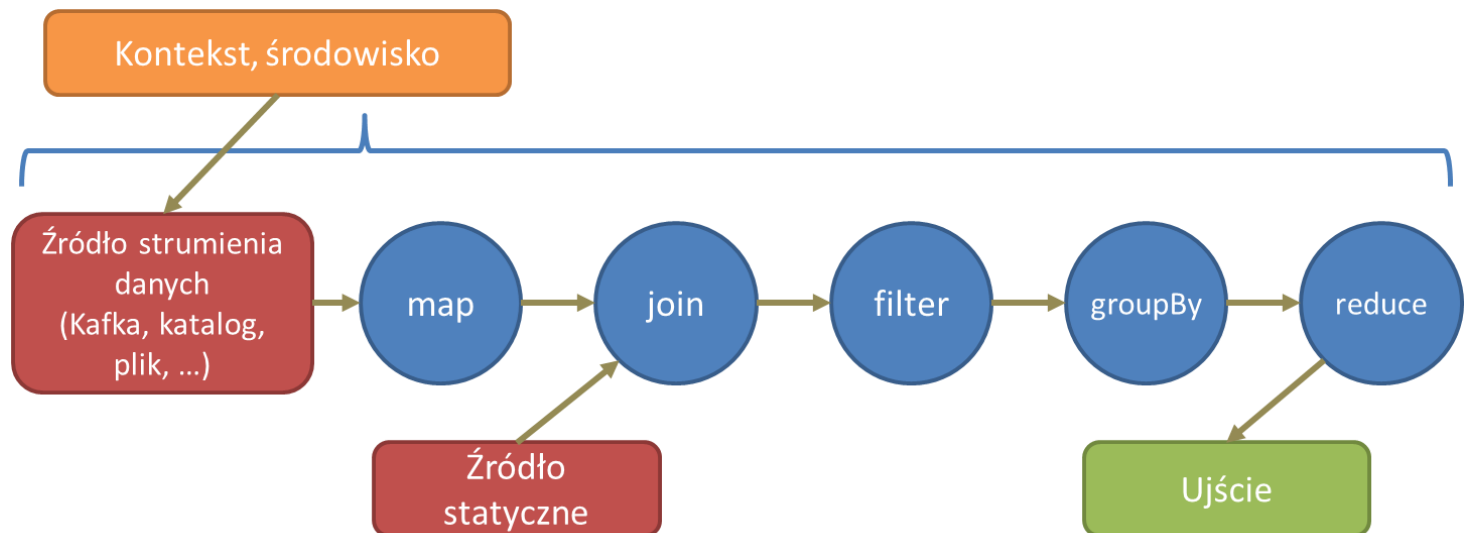
```
val result = orders
    .groupBy($"a")
    .select($"a",
            $"b".count as "cnt")
    .toDataSet[Row]
    .print()
```

```
SELECT a, COUNT(b) as cnt
FROM Orders
GROUP BY a
```

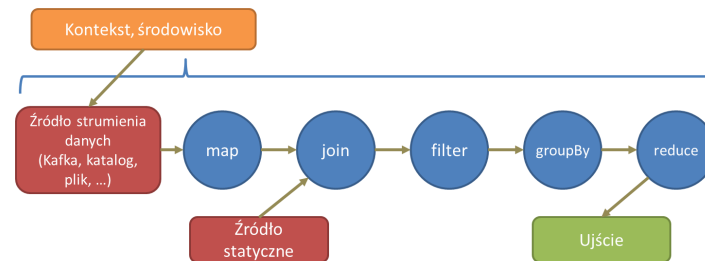

Budowa programu – implementacja

- W wielu systemach, w szczególności korzystających z API na poziomie *DataStream* oraz *Table API*, budowa aplikacji opiera się na pojęciach:
 - kontekstu,
 - źródła,
 - transformacji,
 - ujścia

Kontekst jest obiektem odpowiadającym za połączenie z klastrem silnika przetwarzającego strumienie danych



Przykłady



```
val conf = new
```

```
val ssc = new
```

```
val lines = ssc
```

```
val words = li
```

```
val pairs = wo
```

```
val wordCounts
```

```
wordCounts.pri
```

```
ssc.start()
```

```
ssc.awaitTermi
```

```
val spark = SparkSession.builder
  .appName("StructuredNetworkWordCount")
  .getOrCreate()
```

```
import spark.implicits._
```

```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999).load()
```

```
val words = lines.as[String].flatMap(_.split(" "))
val wordCounts = words.groupBy("value").count()
```

```
val query = wordCounts.writeStream
  .outputMode("complete").format("console")
  .start()
```

```
query.awaitTermination()
```

Źródła i ich typy

- Liczby obsługiwanych źródeł różnią się w zależności od
 - silnika (i jego wersji)
 - typu wykorzystywanego API
- Podstawowe typy źródeł
 - **testowe**, np.: konsola, sockety TCP
 - **plikowe**, np.: pliki, katalogi
 - **systemy kolejkowe**, np.: Apache Kafka, RabbitMQ
 - **inne systemy przetwarzające strumienie danych**, np.: Apache Storm, Apache Samsa, Amazon Kinesis
 - **bazy danych**, a w szczególności mechanizmy **CDC**, np.: Oracle, MongoDB, Neo4j
 - inne...
- W kontekście budowy systemów wiarygodnych źródła dzielimy na
 - **proste** – dostarczające podstawową funkcjonalność, utrudniającą uzyskanie wysokich poziomów gwarancji
 - **wiarygodne** – dostarczające wystarczającej funkcjonalności...

Źródła można także podzielić na:

- wbudowane – dostępne w silniku
- użytkownika – wymagające utworzenia tzw. konektora

Przykładowo patrz:

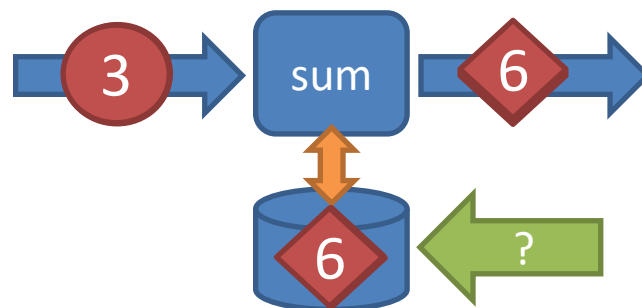
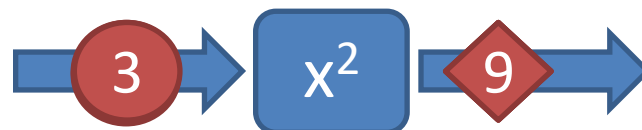
<https://nightlies.apache.org/flink/flink-docs-master/docs/connectors/table/overview/>
<https://www.confluent.io/product/connectors/>

Transformacje i ich typy

- Transformacje służące do implementacji przetwarzania strumieni danych z wykorzystaniem *DataStream* i *Table API* można podzielić na wiele kryteriów
 - Utrzymywanie stanu
 - bezstanowe (*stateless*)
 - DataStream API, np.: `filter`, `map`
 - Table API, np.: `where`, `select`
 - stanowe (*stateful*)
 - wbudowane
 - własne
 - Liczba przetwarzanych strumieni
 - pojedynczy
 - wiele, np.: połączenie, `split`, `union`
 - Typu
 - transformujące
 - agregujące
 - funkcje okna
 - połączenia
 - strumień – tabela
 - strumień – strumień
 - inne

Przetwarzanie stanowe – wprowadzenie

- Co to jest stan?
- Przetwarzanie bezstanowe
 - Cechy
 - Przykłady
- Przetwarzanie stanowe
 - Cechy
 - Przykłady
 - Interaktywne zapytania
 - Jaki strumień (co) jest wynikiem przetwarzania stanu?
 - complete
 - append
 - update



key	value		key	value		key	value
A	1	➡	A	1	➡	A	1
B	2		B	2		B	5
		C	2				

Ujścia

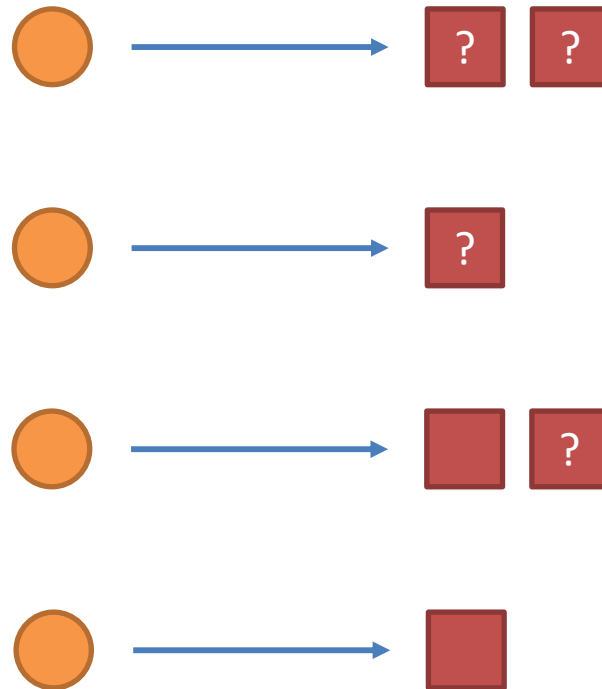
- Liczby obsługiwanych ujść różnią się w zależności od
 - silnika (i jego wersji)
 - typu wykorzystywanego API
- Podstawowe typy ujść
 - **testowe**, np.: konsola, sockety TCP
 - **plikowe (lokalne, rozproszone)**, np.: pliki, katalogi
 - **systemy kolejkowe**, np.: Apache Kafka, RabbitMQ
 - **inne systemy przetwarzające strumienie danych**, np.: Apache Storm, Apache Samsa, Amazon Kinesis
 - **bazy danych**, w szczególności bazy danych NoSQL, np.: Oracle, MongoDB, Neo4j, Elasticsearch
 - inne...
- Kluczowe są możliwości funkcjonalne ujść.
W zależności od przypadku wymagana może być
 - możliwość **modyfikacji** wprowadzonych uprzednio danych
 - skalowalność – obsługa "gęstego" i "nieskończonego" (teoretycznie) strumienia danych wyjściowych

Przykładowo patrz:

<https://nightlies.apache.org/flink/flink-docs-master/docs/connectors/table/overview/>

Gwarancje

- Gwarancje
- Typy gwarancji
 - brak gwarancji
(*no guarantee*)
 - co najwyżej raz
(*at most once*)
 - przynajmniej raz
(*at least once*)
 - dokładnie raz
(*exactly once*)
- System jest tak dobry jak jego najgorszy element



Czas – kilka pytań

- Czy czas podczas przetwarzania strumieni danych ma znaczenie?
- O jakim czasie mówimy?
- W jaki sposób mierzymy czas?
- Czy czas biegnie tylko do przodu?
- Czy dane pojawiają się "na czas"?

Pytania?

Przetwarzanie strumieni danych w systemach Big Data

część 3 – czas

Krzysztof Jankiewicz

Plan

- Czas – dwie (trzy) domeny
- Okna czasowe
- Obsługa zdarzeń nieuporządkowanych
- Wyzwalacze
- Utrzymywanie stanu okien
- Zawartość wyniku

Animacje i część rysunków:

“*Streaming Systems*” by Tyler Akidau, Slava Chernyak, and Reuven Lax (O’Reilly).

Copyright 2018 O’Reilly Media, Inc., 978-1-491-98387-4.

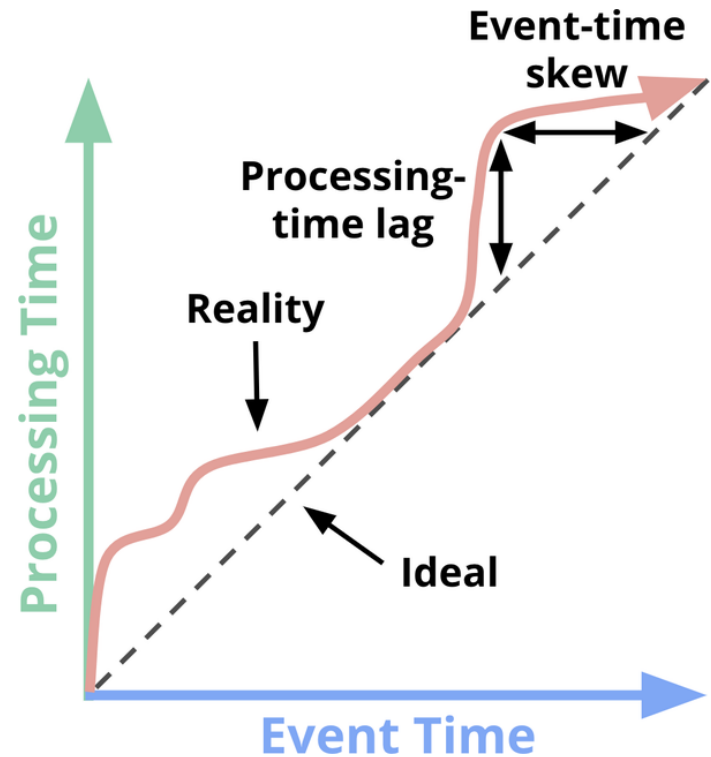
Czas – dwie (trzy) domeny

- Czas przetwarzania (*processing time*)
 - stosowany od systemów I generacji
 - nie pozwala przetwarzać danych
 - zgodnie z momentem ich rzeczywistego powstania
 - historycznych
 - nie wystarczający do wielu przypadków użycia
- Czas zdarzeń (*event time*) – rzeczywisty czas powstania zdarzenia
 - stosowany od systemów III generacji
 - pozwala na implementację architektury Kappa (dane historyczne)
 - powinien uwzględniać możliwe nieuporządkowanie danych
- Czas pozyskania (*ingestion time*) – czas pojawienia się zdarzenia w ogólnie rozumianym systemie
 - porównywalny do czasu zdarzeń (przy założeniu, że zdarzenia docierają do systemu w przybliżeniu w czasie rzeczywistym, bez opóźnień)
 - wymagania dla systemów przetwarzania analogiczne jak dla czasu zdarzeń
 - często nie jest wyodrębniany jako oddzielna domena czasu



Czas przetwarzania a czas zdarzeń

- Świat nie jest idealny – czas przetwarzania \neq czas zdarzeń
 - ograniczenia zasobów
 - oprogramowanie
 - rozproszenie przetwarzania
 - charakterystyka danych
- Opóźnienie w przetwarzaniu (*processing time lag*) – opóźnienie pomiędzy czasem wystąpienia zdarzenia a momentem jego przetwarzania
- Odchylenie czasu zdarzenia (*event-time skew*) – mówi jak daleko od ideału (w stosunku do czasu zdarzenia) jest przetwarzanie danych



Wskaż na powyższym wykresie fragmenty czerwonej linii, które odpowiadają poniższym sytuacjom:

System, którego połączenie ze światem zewnętrznym znacząco ograniczyło przepustowość

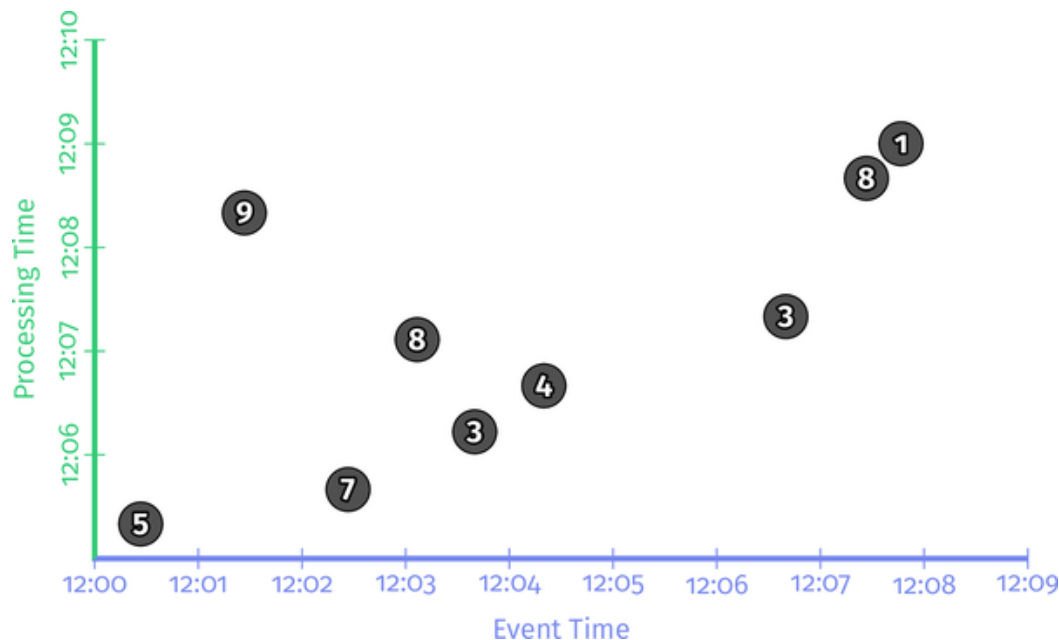
Pasażerowie samolotu, po którego wylądowaniu wyłączają tryb samolotowy w swoich urządzeniach, wysyłając i odbierając w krótkim okresie czasu wszystko, co wydarzyło się przez ostatnich kilka godzin.

Przykład strumienia danych

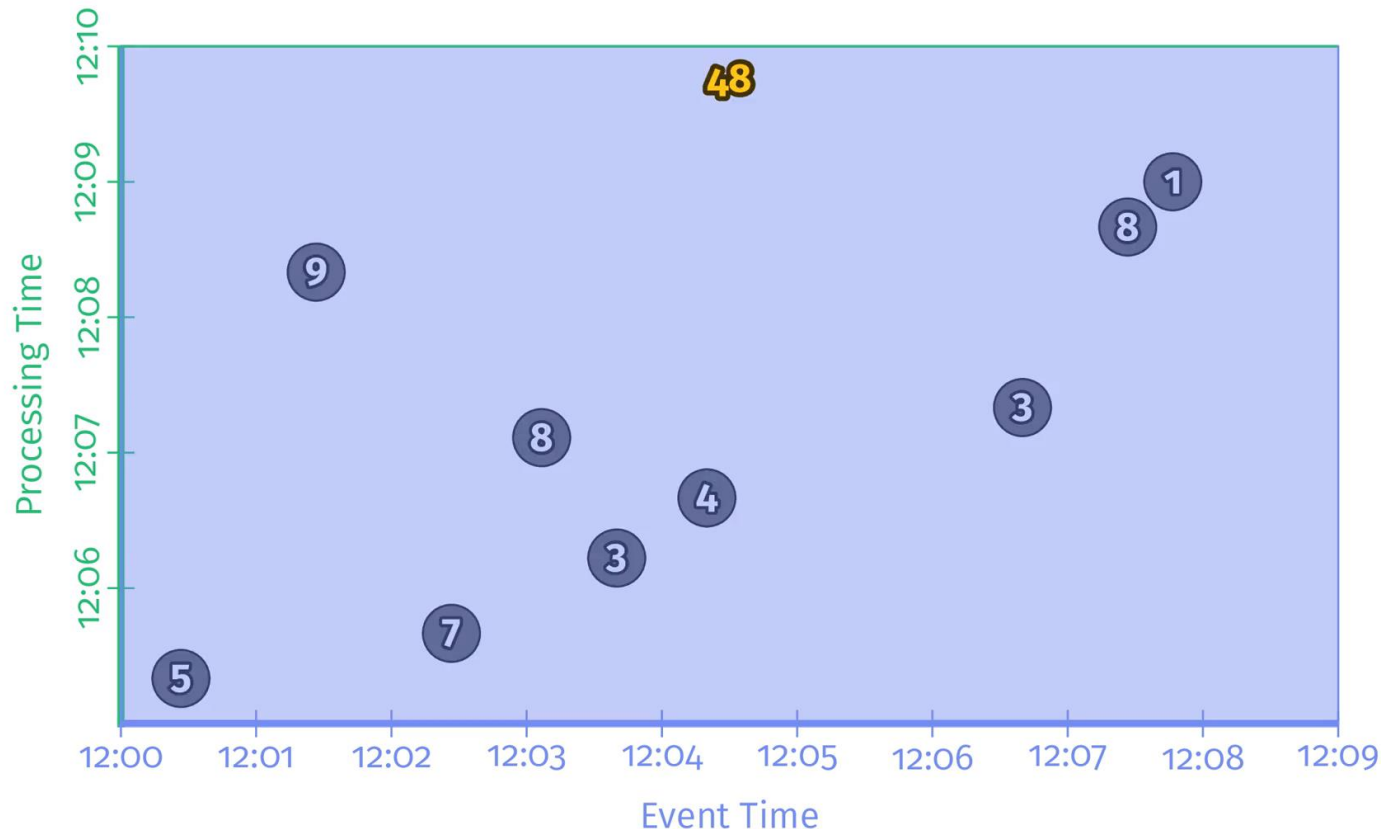
Name	Team	Score	EventTime	ProcTime
Severus Snape	Gryffindor	5	12:00:26	12:05:19
Syriusz Black	Gryffindor	7	12:02:26	12:05:39
Hermiona Granger	Gryffindor	3	12:03:39	12:06:13
Albus Dumbledore	Gryffindor	4	12:04:19	12:06:39
Severus Snape	Gryffindor	8	12:03:06	12:07:06
Rubeus Hagrid	Gryffindor	3	12:06:39	12:07:19
Harry Potter	Gryffindor	9	12:01:26	12:08:19
Minerwa McGonagall	Gryffindor	8	12:07:26	12:08:39
Rubeus Hagrid	Gryffindor	1	12:07:46	12:09:00

Będziemy sumowali
wyniki uzyskiwane
przez poszczególne
drużyny

Upraszczamy nasz
przykład zajmując się
tylko jedną drużyną
(jedną partycją
strumienia)



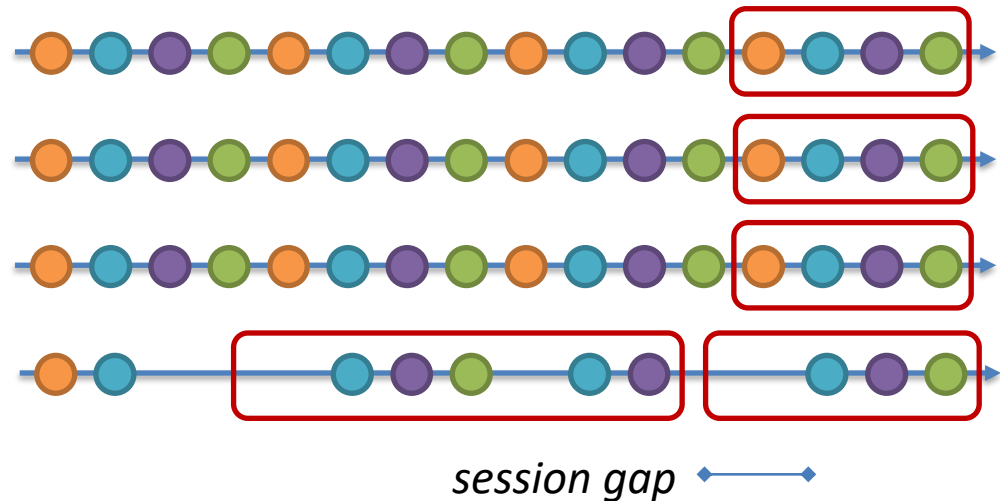
Przetwarzanie wsadowe na całości danych



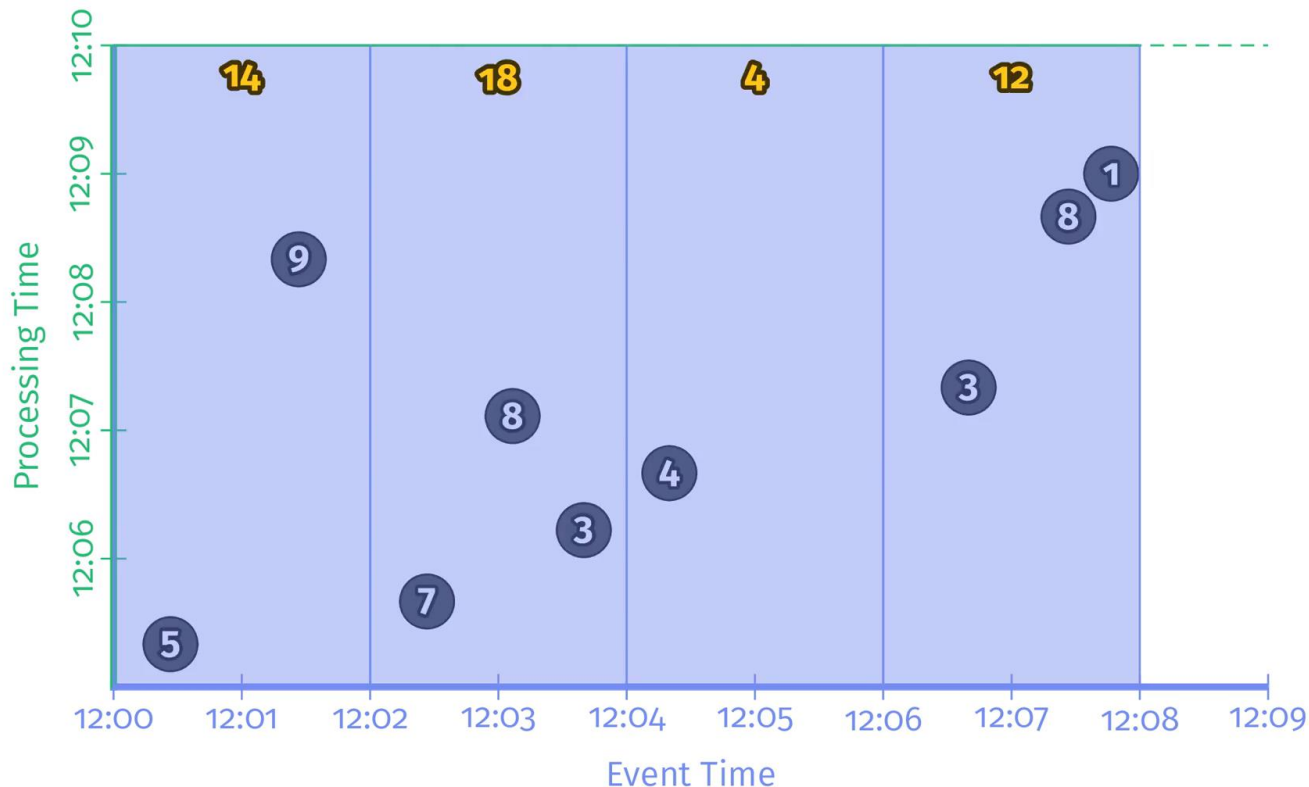
- Nieodłączny element czasu związany ze strumieniami danych powoduje, że w większości przypadków chcemy, aby otrzymywane wyniki były przypisane do określonych interwałów czasu – okien
- Okna mogą funkcjonować w różnych domenach czasu, choć głównie w domenie czasu zdarzeń

Okna

- Podział strumienia na podzbiory
- Różne typy i definicje przydziału zdarzeń do podzbiorów
- Oparte o czas (bardzo rzadko o liczbę zdarzeń)
- Typy okien
 - *Tumbling*
 - *Hopping*
 - *Sliding*
 - *Session*
 - Globalne
 - Własne
- Uproszczony podział
 - Stałe (*fixed*) – *tumbling*
 - Przesuwne (*sliding*) – *sliding, hopping*
 - Sesyjne (*sessions*) – *session*



Przetwarzanie wsadowe okien



Okna

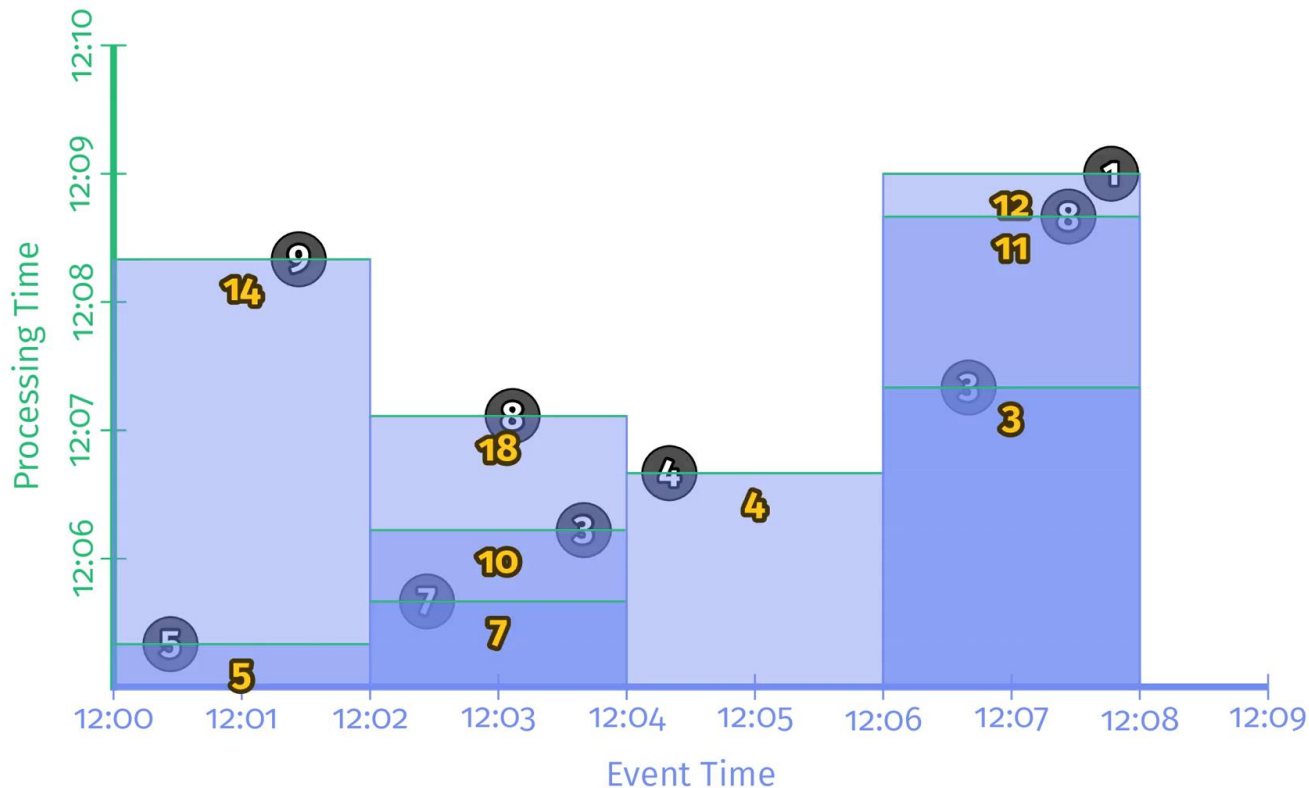
- oparte o etykiety czasowe zdarzeń
- o długości 2 minut

- W przypadku nieskończonych strumieni danych, nie jest praktycznym oczekiwanie na ich zakończenie w celu uzyskanie ostatecznego wyniku – w niektórych ;) przypadkach opóźnienie może nie być akceptowalne
- Rozwiązaniem są wyzwalacze...

Wyzwalacze

- Wyzwalacza definiują **kiedy** wynik przetwarzania okna ma być wygenerowany (ma zostać zmaterializowany)
- Każdy wynik wygenerowany dla określonego okna określany bywa taflą (*pane*)
- Wyzwalacze funkcjonują w domenie czasu przetwarzania (choć na decyzję mogą wpływać znaczniki *watermark* funkcjonujące w domenie czasu zdarzeń)
- Logika działania wyzwalaczy może być bardzo różna, jednak zazwyczaj można je zaliczyć do:
 - **wyzwalaczy powtarzalnej aktualizacji** (*repeated update triggers*) – okresowo generują tafle okna, którego zawartość ulega zmianie
 - **wyzwalaczy kompletności** (*completeness triggers*) – materializują wynik jedynie wówczas, kiedy okno można uznać za kompletne
 - będących kombinacją obu powyższych klas

Wyzwalacze powtarzalnej aktualizacji



Okna

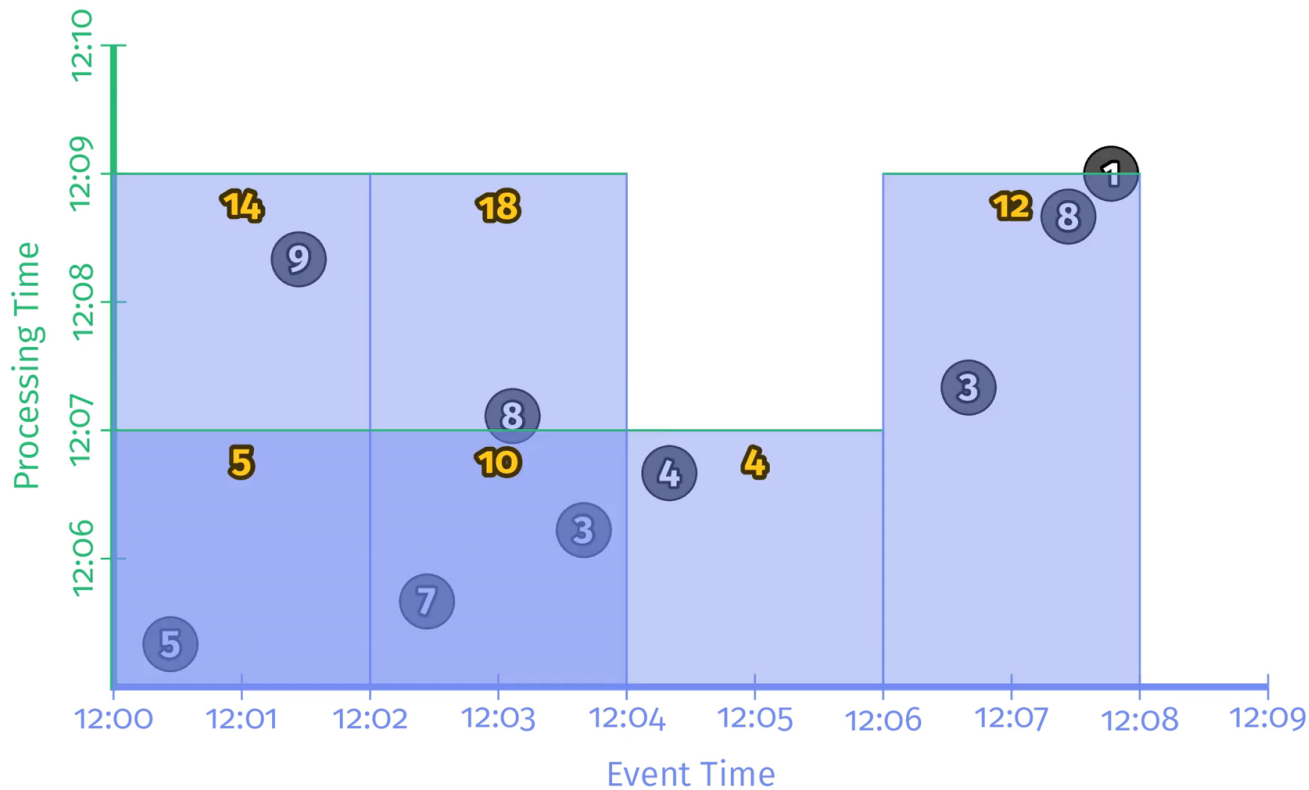
- oparte o etykiety czasowe zdarzeń
- o długości 2 minut

Wyzwalacz

- uruchamiany dla każdego zdarzenia

- Wiele taflí dla każdego okna
- Dane na wyjściu są zawsze "na bieżąco"
- Wyniki generowane są bardzo często (co w przypadku dużej ilości danych może stwarzać problemy wydajnościowe)
- Rozwiązaniem są – wyzwalacze powtarzalnej aktualizacji działające z opóźnieniem

Wyzwalacze z wyrównanym opóźnieniem



Okna

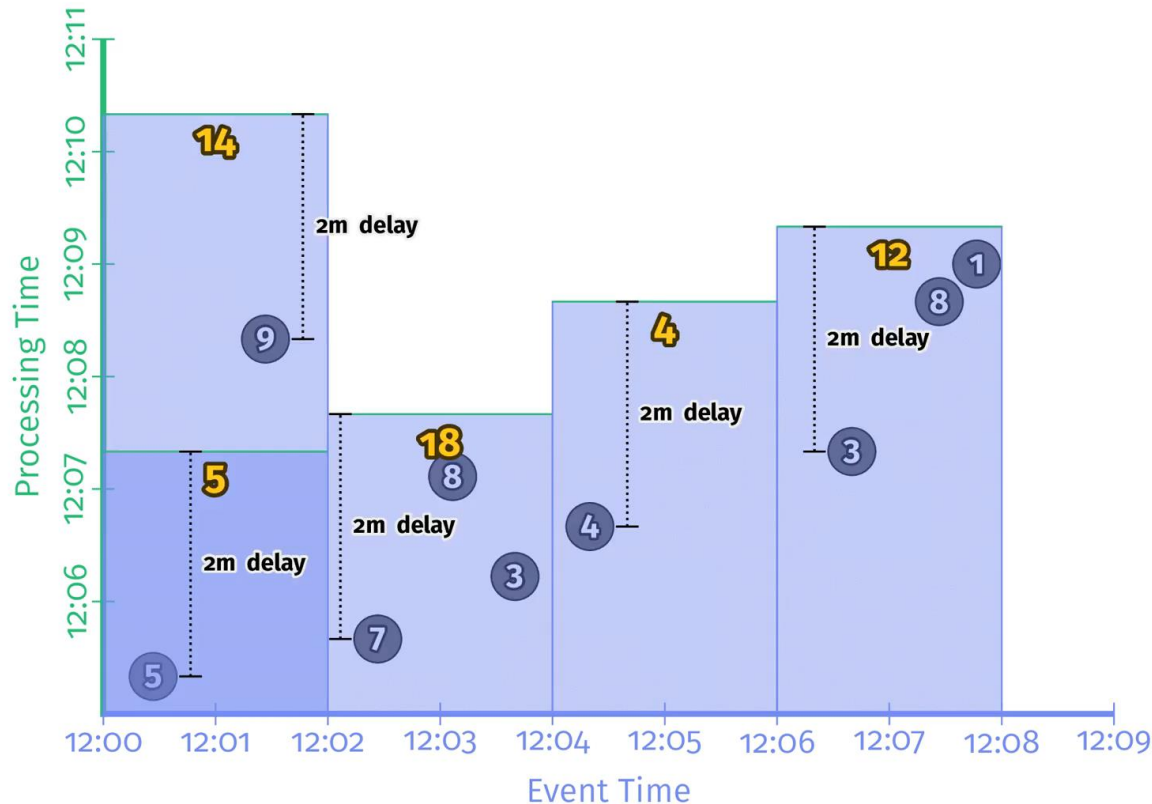
- oparte o etykiety czasowe zdarzeń
- o długości 2 minut

Wyzwalacz

- uruchamiany z wyrównanym opóźnieniem o wielkości 2 minut

- Rozwiązanie, z którym mamy do czynienia w niektórych systemach mikro-wsadowych
 - *Spark Streaming* – wielkości opóźnienia równe
 - *Spark Structured Streaming* – wielkości opóźnienia zależne od obciążenia lub równe
- Przewidywalne i regularne generowanie wyniku
- Generowanie wyników w tym samym czasie dla wszystkich okien (wszystkich partycji) prowadzi do pików obciążenia
- Rozwiązaniem są wyzwalacze z niewyrównanym opóźnieniem

Wyzwalacze z niewyrównanym opóźnieniem



Okna

- oparte o etykiety czasowe zdarzeń
- o długości 2 minut

Wyzwalacz

- uruchamiany z niewyrównanym opóźnieniem o wielkości 2 minut

- Zalety wyzwalaczy z niewyrównanym opóźnieniem dla systemów dużej skali
 - Bardziej równomierne generowanie wyników (tafli) w czasie
 - Generowanie mniejszej liczby tafli
- Korzystanie z wyzwalaczy działających z opóźnieniem skutkuje tym, że
 - dane wynikowe nie są utrzymywane "na bieżąco"
 - poprawność danych wynikowych (aktualność) nie jest w każdej chwili gwarantowana, a czas jej uzyskania nie jest do końca określony

Znaczniki *watermark*

- **Poprawność** danych wynikowych jest **gwarantowana** dla okien, które przetworzyły wszystkie swoje dane – **okien kompletnych**
- **Wiedza** dotycząca kompletności okien może być wykorzystania przez **wyzwalacze kompletności**
- Jej **źródłem** w systemach przetwarzających strumienie danych mogą być **znaczniki *watermark***
- Znaczniki *watermark* funkcjonują w domenie **czasu zdarzeń**
- Znaczniki *watermark* nigdy **nie cofają się** w czasie – ich wartości monotonicznie **rosną**
- Konceptyjnie znaczniki *watermark* można potraktować jako **funkcję**

$$f(P) \rightarrow E$$

gdzie:

P – jest czasem przetwarzania, w którym wyznaczany jest znacznik *watermark*

E – jest czasem w domenie czasu zdarzeń wskazującym, że wszystkie zdarzenia mające czas zdarzeń mniejszy niż E zostały już dostarczone

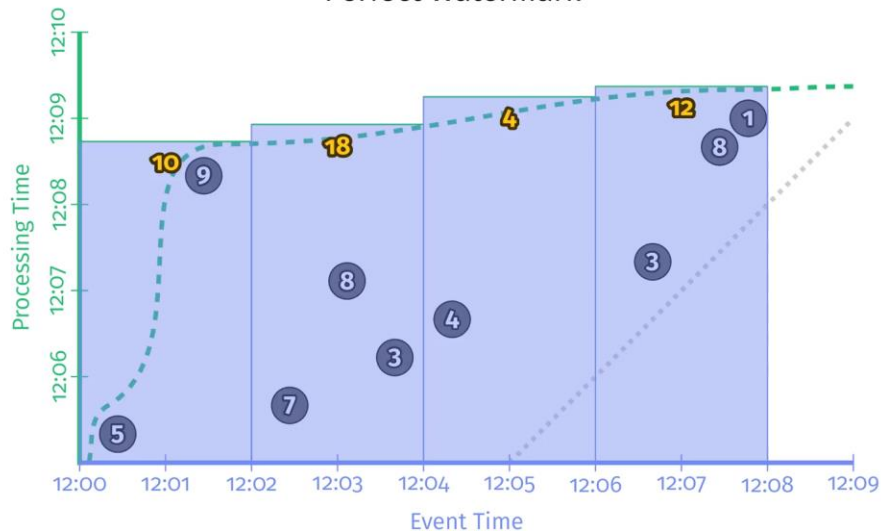
Typy znaczników *watermark*



- **Dokładne** (*perfect watermarks*) – gwarantowane
 - dostępne w sytuacji, gdy mamy dokładną (pełną) wiedzę na temat danych wejściowych
 - dane, które nie spełniają deklaracji znacznika *watermark* zdarzenia spóźnione (*late data*) – nie występują
- **Szacunkowe** (*heuristic watermarks*)
 - stosowane w znaczącej większości przypadków
 - wykorzystywane wówczas, gdy dokładna wiedza na temat danych wejściowych jest nieosiągalna
 - tworzone są na podstawie charakterystyki danych źródłowych a także wykorzystywanych do ich przetwarzania (wcześniejszych) systemów
 - nawet przy najlepszych oszacowaniach, dane spóźnione mogą pojawić się w strumieniu wejściowym

Wyzwalacze kompletności

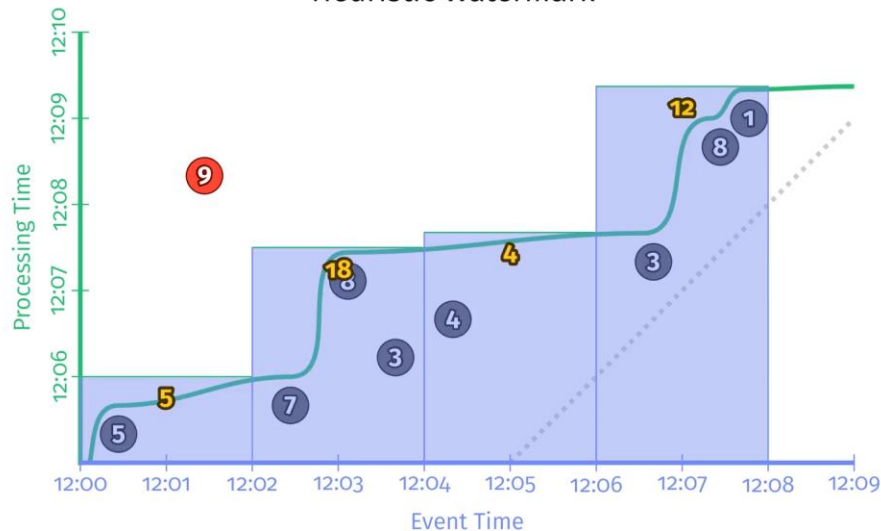
- Mając znaczniki *watermark* możemy wykorzystać wyzwalacze, które na nich bazują.
- Wyzwalacz kompletności materializuje wynik dla okna (tworzy tabelę) w momencie, gdy znacznik *watermark* przechodzi przez koniec okna



Perfect Watermark



Perfect watermark: 
Ideal watermark: 

Heuristic Watermark



Heuristic watermark: 
Ideal watermark: 

Znaczniki *watermark* wykorzystywane są także przy połączeniach strumieni danych

"Dylematy" znaczników *watermark*

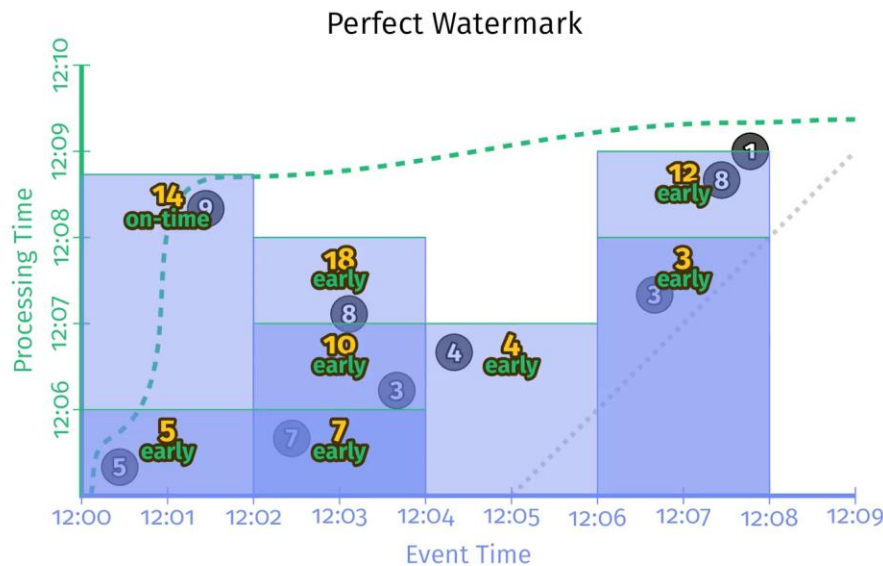
- W kontekście korzystania z wyzwalaczy kompletności znaczniki *watermark* mogą być generowane:
 - zbyt późno
 - patrz: dokładny *watermark* na naszym przykładzie – wyniki z
 - zdarzenia, które znacząco odbiegają od sytuacji idealnej (czas przetwarzania = czas zdarzeń) wstrzymują generowanie wyników dla wielu okien
 - są dobre z punktu widzenia kompletności wyników
 - bardzo negatywnie wpływają na opóźnienia
 - zbyt szybko
 - tylko w przypadku szacunkowych znaczników *watermark*
 - prowadzą do niepoprawnych rezultatów (nie obejmujących zdarzeń spóźnionych)



Jak rozwiązać ten problem?

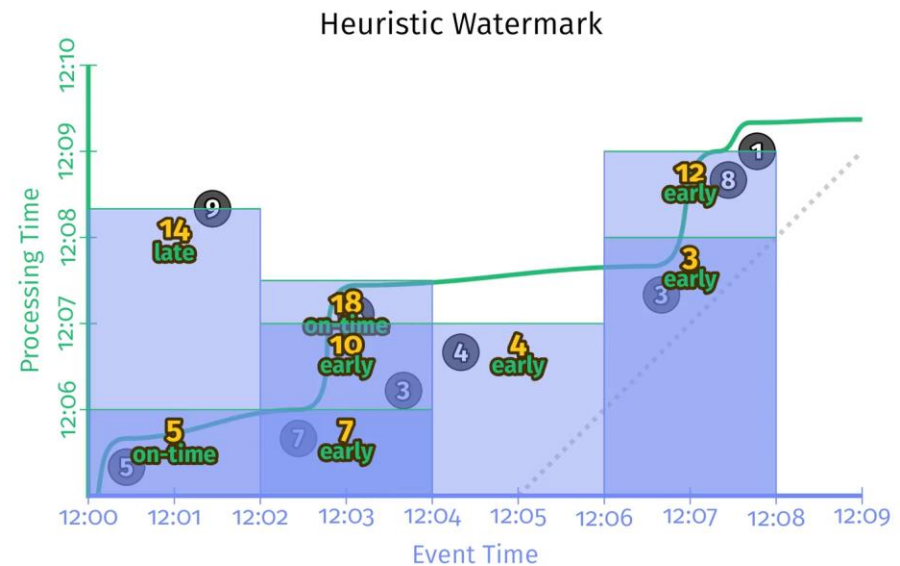
Złożone wyzwacacze



- Rozwiązaniem może być wykorzystanie złożonych typów wyzwacaczy
- Jedna z kategorii takich wyzwacaczy określana jest mianem *wcześnie/na czas/późno (Early/On-Time/Late – EOTL)*. Jest ona kombinacją trzech typów wyzwacaczy
 - *wcześnie* – przed znacznikiem *watermark* generowane są okresowo wyniki (np. w oparciu o mechanizm wyzwacaczy z wyrównanym opóźnieniem) – zero lub więcej *wczesnych* tafli
 - *na czas* – znacznik *watermark* generuje kolejny wynik (w oparciu o wyzwacacz kompletności) – co najwyżej jedna tafa *na czas*
 - *późno* – po znaczniku *watermark* generowane są ponownie okresowo wyniki (np. wyzwacacz uruchamiany dla każdego zdarzenia) – zero lub więcej *późnych* tafli

Wyzwalacz wcześnie/na czas/późno (*EOTL*)



Perfect watermark: 
Ideal watermark: 



Heuristic watermark: 
Ideal watermark: 

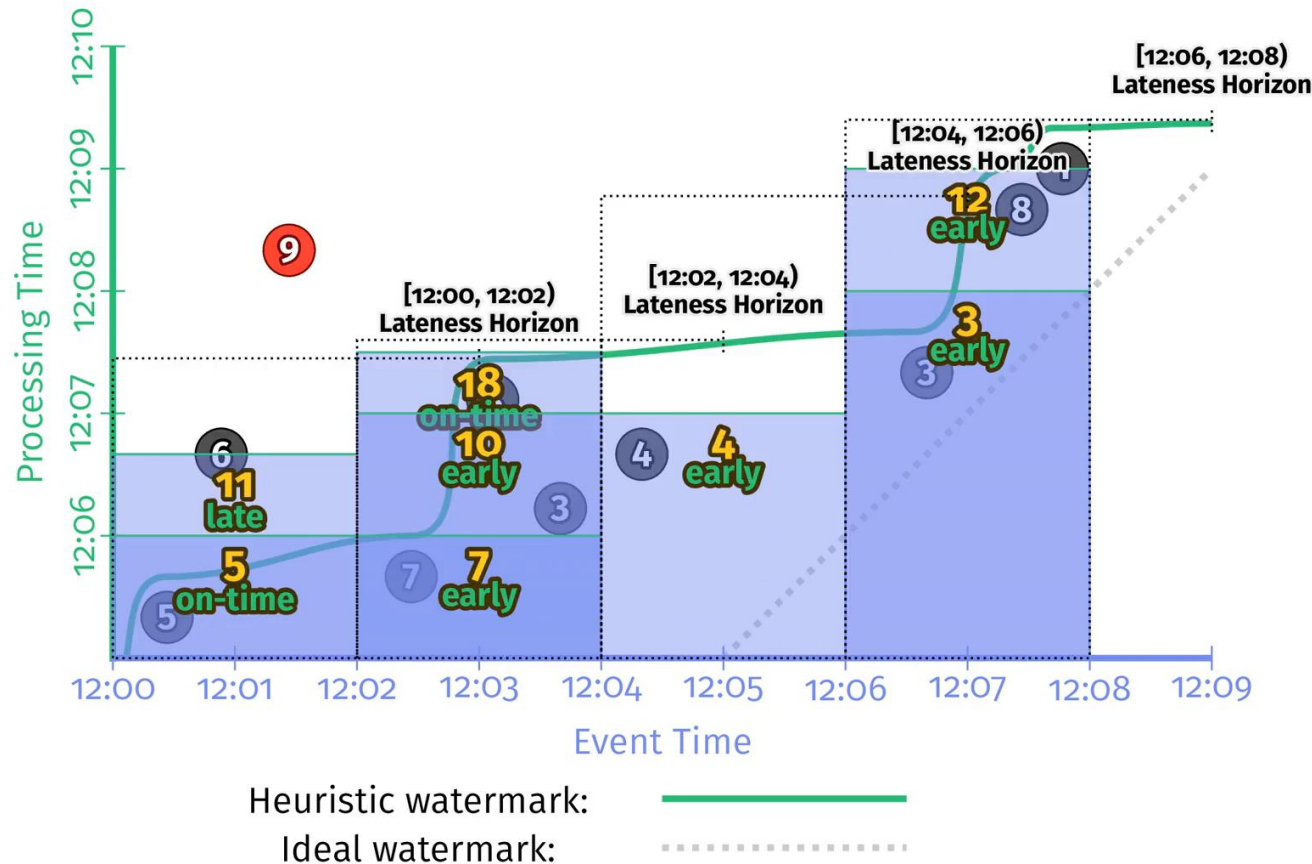
- Wyniki generowane są w sposób przybliżony dla obu typów znaczników *watermark* (wyzwalacz szacunkowy nie może być dramatycznie "zbyt szybki")
- Różnica pomiędzy typami znaczników *watermark* polega na możliwości usunięcia stanu okna
 - dokładny – dane spóźnione nigdy się nie pojawią – stan okna można usunąć
 - szacunkowy – dane spóźnione mogą się pojawić – stan okna trzeba nadal utrzymywać

Wcześnie –
wyzwalacz z
wyrównanym
opóźnieniem
co 1 minuta

Utrzymywanie stanu okien

- W praktyce nie ma możliwości przechowywania stanu okien w nieskończoność
- Konieczne jest określenie **horyzontu** dopuszczalnego **opóźnienia**, wyznaczającego dopuszczalny zakres opóźnienia danych
- Horyzont taki **pozwała na usunięcie stanu** okna
- Zdarzenia, które go **przekroczą** mogą zostać
 - **usunięte**
 - obsłużone w inny, nie oparty na stanie okna sposób (alternatywa)
- Dzięki takiemu rozwiązaniu zasoby wykorzystywane przez dane, którymi "nie jesteśmy zainteresowani" mogą zostać zwolnione

Obsługa horyzontu dopuszczalnego opóźnienia



Okna

- oparte o etykiety czasowe zdarzeń
- o długości 2 minut

Wyzwalacz

- *EOTL*

Horyzont opóźnienia

- 1 minuta opóźnienia w stosunku do znacznika *watermark*

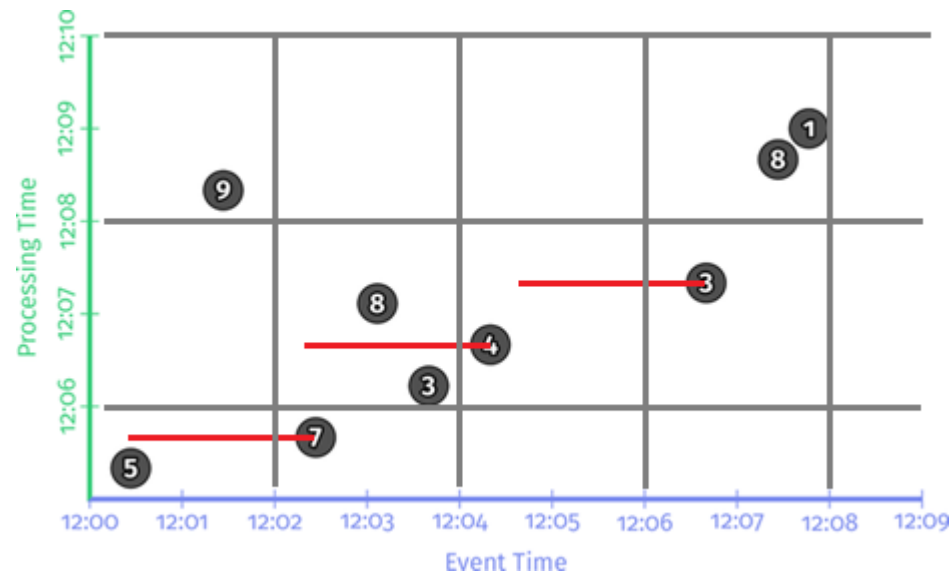
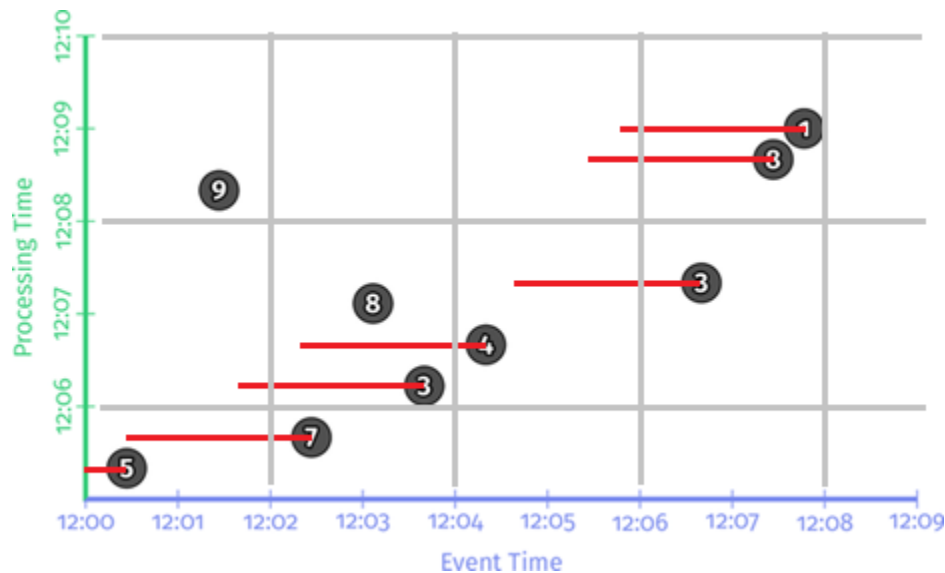
- Horyzont dopuszczalnego opóźnienia nie jest potrzebny gdy:
 - korzystamy z dokładnego znacznika *watermark*
 - dokonujemy obliczeń niezwiązanych z oknami – np. agregacji dla druzyn w całym okresie czasu – zakładamy w takim przypadku, że liczba kluczy jest ograniczona (w odróżnieniu od liczby okien)

Niskie (*low*) i wysokie (*high*) watermark

- Znaczniki *watermark* wyznaczane są w oparciu o zaobserwowane znaczniki czasu zdarzeń
 - **niskie** – na **najstarszych** znacznikach
 - **wysokie** – na **najnowszych** znacznikach

Przykład:

- szacunkowy *watermark* – minus 2 minuty od zaobserwowanego znacznika



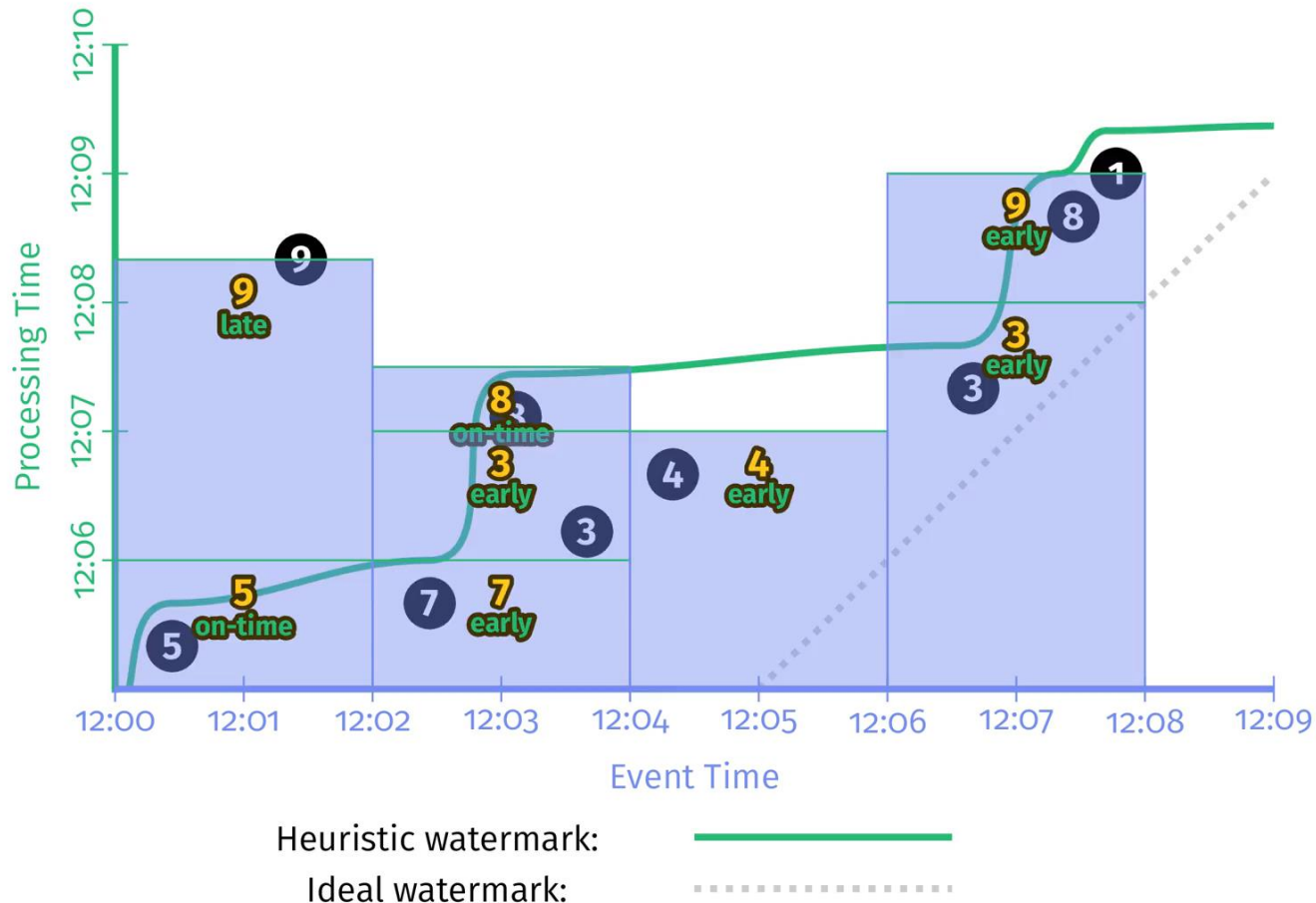
- Wysoki znacznik *watermark* stosowany jest przykładowo w *Spark Structured Streaming*
- Niskie znaczniki *watermark* stawiają na poprawność wyniku
- Wysokie znaczniki *watermark* koncentrują się na małym opóźnieniu (mając większą tendencję do występowania zdarzeń spóźnionych)

Zawartość wyniku (tafli)

- Wiemy już w jaki sposób zarządzamy
 - czasem generowania wyników (tafli)
 - czasem usuwania stanów okien
- Pozostaje pytanie: co powinno być zawarte w kolejnych wynikach dla danego okna?
- Trzy podejścia:
 - porzucanie
 - każda tafla obejmuje tylko dane, które pojawiły się po wygenerowaniu poprzedniej tafli
 - wynik każdej tafli jest porzucany – nie jest wykorzystywany do kolejnych obliczeń
 - przydatne gdy odbiorca wyników dokonuje samodzielnie ich agregacji
 - akumulacja
 - tafla obejmuje wszystkie dane jakie pojawiły się do tej pory
 - wynik każdej tafli jest wykorzystywany do kontynuowania obliczeń dla tafli następnych
 - przydatne gdy odbiorca zastępuje wartości zaobserwowane do tej pory
 - akumulacja i wycofanie
 - tafla zawiera zarówno wynik obejmujący akumulację wartości jak i wycofane wartości poprzednie

Zawartość wyniku (tafli) – porzucanie

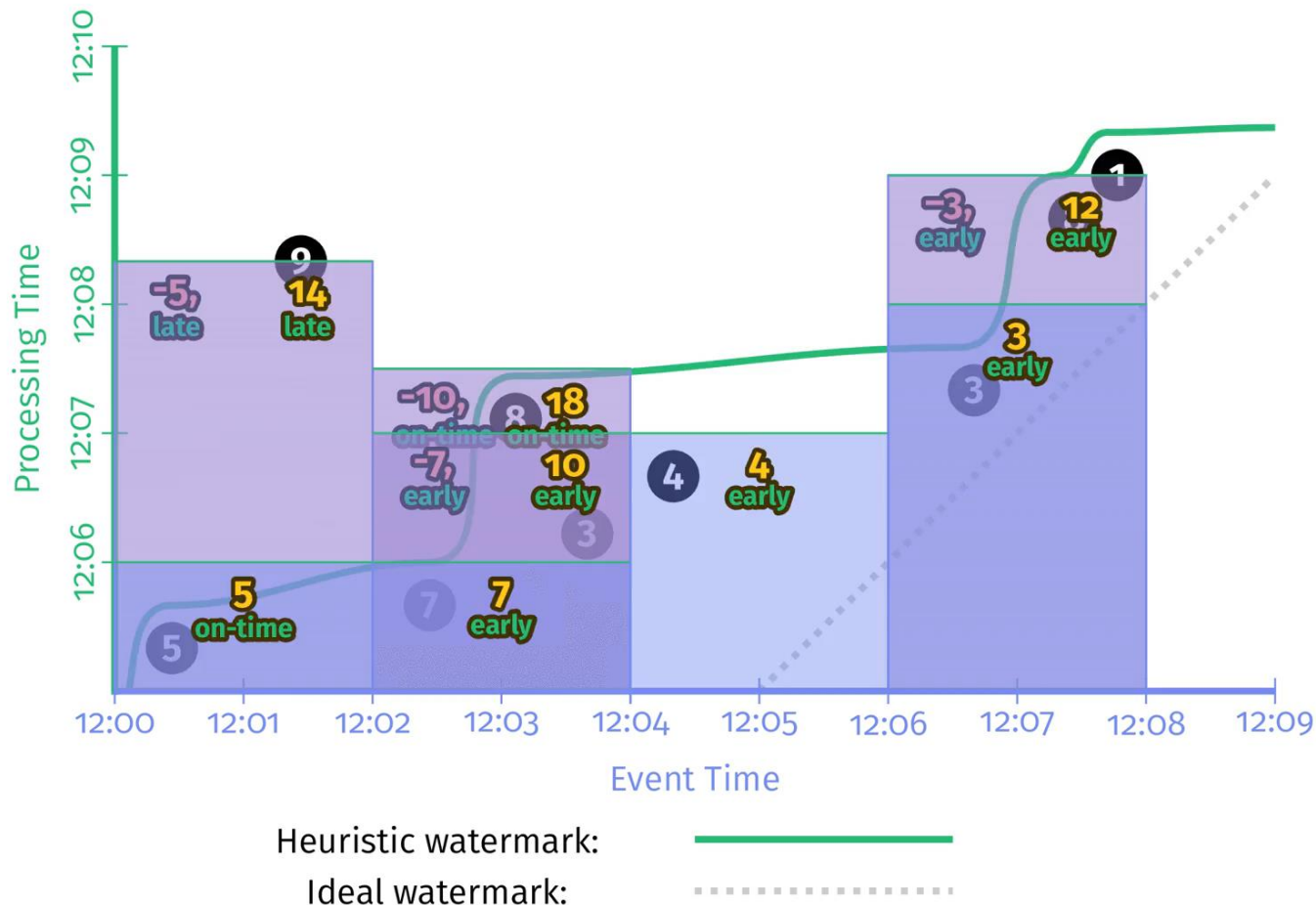
- W założeniu, korzystamy z wyzwalacza *EOTL*.



Zawartość wyniku (tafli)

akumulacja i wycofanie

- W założeniu, korzystamy z wyzwalacza *EOTL*.



Zawartość wyniku (tafli)

akumulacja i wycofanie

- Daje duże możliwości w kontekście dalszej obsługi
 - zmiana sposobu grupowania
 - obsługa dynamicznych okien (np. sesyjnych)
- Stosowane bardzo często w systemach przetwarzania strumieni danych
- W szczególności widoczne w wyższych poziomach abstrakcji (np. SQL)

Przykład

- Wynikiem przetwarzania strumieni jest to kto dla jakiej drużyny zdobył ostatnio punkty:
`select character, last(team)`
`from scores`
`group by character`
- Na podstawie wyniku odbiorca chce utrzymywać liczbę postaci grających dla każdej z drużyn
- Rozważ obsługę trzech typów wyników przetwarzania strumieni:
 - porzucanie
 - akumulacja
 - akumulacja i wycofanie

Pytania?

Przetwarzanie strumieni danych w systemach Big Data

część 4 – dualizm strumieni i tabel, SQL

Krzysztof Jankiewicz

Plan

- Wyższe poziomy abstrakcji
- Narracje
- Fundamenty
- Perspektywy materializowane
- Dualizm strumieniowo tabelowy
- Typy strumieni
- Znaczenie metadanych

Wyższe poziomy abstrakcji

- Przetwarzanie strumieni danych – podstawowy poziom abstrakcji
 - traktowanych jako nieskończona kolekcja (sekwencja) zdarzeń
 - za pomocą API przypominającego przetwarzanie kolekcji
- Wyższe poziomy abstrakcji z reguły opierają się na podejściu "relacyjnym"
 - strumień traktowany jest jako strumień poleceń DML
 - operacji wstawiania danych – nowe dane
 - operacji aktualizacji danych – najnowsze wersje danych
 - przykłady
 - TableAPI – oparty na metodach odpowiadających klauzulom SQL
 - SQL – całkowicie oparty na "variantach" poleceń SQL, w których występują zarówno ograniczenia jak i rozszerzenia standardu SQL (wynikające głównie z natury danych)

Narracje

- Wyższe poziomy abstrakcji funkcjonują na poziomie dwóch światów
 - strumieni
 - tabel
- Stosowane narracje
 - Apache Kafka Streams
 - Strumienie jako *record stream* – każdy wiersz to nowa dana
 - Tabele jako *changelog stream* – aktualizacje na poziomie klucza
 - Apache Flink
 - tabele w bazie danych utrzymywane są na podstawie strumienia poleceń DML - *changelog stream*
 - perspektywy materializowane definiowane są za pomocą poleceń SQL. Ich aktualizacja wymaga ciągłej analizy (wykonywania zapytania) i przetwarzania strumienia zmian z tabel bazowych
 - zmiany w tabelach tworzą strumień poleceń DML

Fundamenty

- Mechanizmy łączące tabele i strumienie zmian nie powstały wraz systemami przetwarzania strumieni danych
- Dzienniki powtórzeń (*redo logs*) – przykłady:
 - Relacyjne bazy danych
 - HDFS (*EditLog*)
 - Redis (*append-only file*)
- Mechanizmy replikacji danych
- Mechanizmy CDC (*change data capture*)
 - Typy
 - Log-based CDC
 - Trigger-based CDC
 - ETL-based CDC
 - API-based CDC
 - Przykłady implementacji
 - Oracle CDC - Oracle GoldenGate
 - MongoDB CDC - MongoDB Change Streams
 - Microsoft CDC - Microsoft SQL Server CDC
 - Debezium (np. dla bazy danych PostgreSQL, Oracle, MySQL, Cassandra, MongoDB)

Perspektywy materializowane

- Perspektywa – polecenie SQL
- Perspektywa materializowana (MV)
 - polecenie SQL
 - zmaterializowany wynik – tabela
- Zmiany w tabelach bazowych => nieaktualna perspektywa materializowana
- Sposoby odświeżania
 - pełne (COMPLETE) – zapytanie SQL odbudowuje pełną zawartość MV
 - przyrostowe (FAST) – aktualizuje fragmenty MV na podstawie zmian w tabelach bazowych
- Odświeżanie przyrostowe wymaga wiedzy na temat zmian w tabelach bazowych np. logów perspektyw materializowanych
- Relacje pomiędzy tabelami a strumieniami
 - Log MV – strumień utrzymywany na podstawie zmian w tabeli
 - MV – tabela utrzymywana na podstawie strumienia zmian

Oracle – log perspektywy materializowanej

```
CREATE TABLE example_table (  
  id NUMBER PRIMARY KEY,  
  name VARCHAR2(50),  
  age NUMBER  
);
```

```
INSERT INTO example_table (id, name,  
VALUES (1, 'John', 25);
```

```
INSERT INTO example_table (id, name,  
VALUES (2, 'Jane', 30);
```

```
INSERT INTO
```

```
SELECT * FROM MLOG$_example_table;
```

```
UPDATE example_table  
SET age = 26 WHERE id = 1;
```

```
DELETE FROM example_table  
WHERE id = 3;
```

```
INSERT INTO example_table  
(id, name, age)  
VALUES (4, 'Kate', 35);
```

ID	NAME	AGE	M_ROW\$\$	SNAPTIME\$\$	DMLTYPE\$\$	OLD_NEW\$\$	CHANGE_VECTOR\$\$	XID\$\$
1	John	25	AKJBV6ADCAAAACUAAA	01-JAN-00	U	U	08	8974266703940797171
3	Bob	40	AKJBV6ADCAAAACUAAC	01-JAN-00	D	O	00	8974266703940797171
4	Kate	35	AKJBV6ADCAAAACWAAA	01-JAN-00	I	N	FE	8974266703940797171

Dualizm strumieniowo tabelowy

- Domyślnie strumień (bez dodatkowych metadanych) może być traktowany jako strumień operacji wstawienia (*record stream*)

taxi	godzina	dzielnica	akcja
taxi1	18:00	Wilda	start
taxi2	18:10	Łazarz	start
taxi1	18:15	Piątkowo	stop
taxi3	18:20	Jeżyce	start

- Na podstawie tego strumienia można utrzymywać tabele np.:

- ostatni stan
- liczba akcji

taxi	ostatni stan
taxi1	wolna
taxi2	zajęta
taxi3	zajęta

taxi	liczba akcji
taxi1	2
taxi2	1
taxi3	1

- Tabele

- utrzymują stan (*stateful*),
- często są wynikiem agregacji

- Zmiany w tabelach mogą być źródłem strumieni.

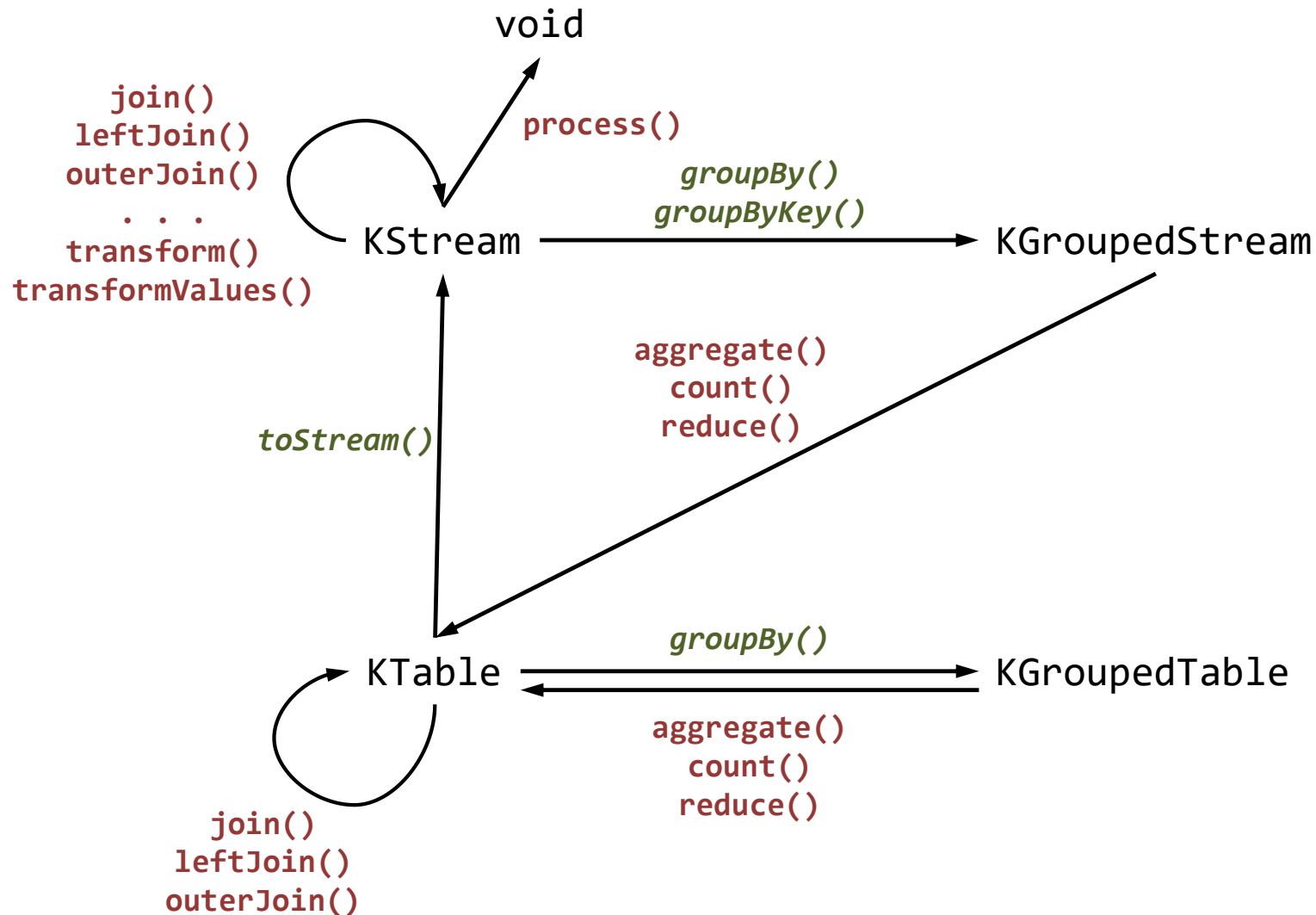
Strumienie takie reprezentują zmiany (*changelog stream*) – to oznacza, że ich interpretacja podczas przetwarzania takiego strumienia powinna być inna

Kafka a tabele

- Apache Kafka posiada specjalny scalony typ tematu (*compacted topics*), który utrzymuje pojedynczą wartość dla danego klucza
- Apache Kafka może implementować takie tematy przy wykorzystaniu wbudowanej bazy danych *in-memory* RocksDB opartej na modelu klucz-wartość.

```
kafka-topics.sh --create \  
  --bootstrap-server ${CLUSTER_NAME}-w-0:9092 \  
  --replication-factor 2 --partitions 3 --topic kafka-compacted \  
  --config cleanup.policy=compact
```

Kafka Streams



Kafka – ksqlDB

```
CREATE STREAM scoreEvents (house VARCHAR, character VARCHAR,  
                           score INT, ts VARCHAR)  
  WITH (kafka_topic='kafka-input', value_format='json',  
        partitions=3);
```


```
CREATE TABLE latest_scores_tab AS  
select house, LATEST_BY_OFFSET(character) character,  
          LATEST_BY_OFFSET(score) score,  
          LATEST_BY_OFFSET(ts) ts  
from scoreEvents  
group by house
```


Flink


```
%flink.sql
CREATE OR REPLACE TEMPORARY TABLE source_table (
  `code` VARCHAR(10),
  `value` INT,
  `ts` TIMESTAMP(3),
  `pt` AS PROCTIME(),
  WATERMARK FOR `ts` AS `ts` - INTERVAL '1' SECOND
)
WITH (
  'connector' = 'faker',
  'number-of-rows' = '20',
  'rows-per-second' = '1',
  'fields.code.expression' = '#{number.numberBetween ''51'', ''54''}',
  'fields.value.expression' = '#{number.numberBetween ''1'', ''9''}',
  'fields.ts.expression' = '#{date.next ''2'' ''SECONDS''}';
);
```


FLINK JOB FINISHED


```
%flink.sql
SELECT `code`, sum(`value`) as sum_val, count(*) as how_many
FROM source_table
GROUP BY `code`;
```





















settings



code	sum_val	how_many	
52	66	17	
53	9	3	

Typy strumieni

- Znamy już dwa typy strumieni *record* i *changelog*
- Operując na mechanizmach znanych z Apache Flink powiedzielibyśmy, o dwóch typach strumieni
 - **Strumień tylko wstawiający (*append-only stream*)** – dynamiczna tabela obsługiwana tylko i wyłącznie za pomocą zmian INSERT odwzorowywana jest na strumień składający się z tylko wstawianych wierszy.
 - **Strumień z aktualizacjami (*upsert stream*)** – zawiera dwa typy komunikatów:
 - **aktualizujące** (*upsert messages*) oraz
 - **usuwające** (*delete messages*).
 - Dynamiczna tabela konwertowana na strumień ze aktualizacjami wymaga unikalnego klucza. W takim przypadku
 - zmiany INSERT oraz UPDATE generują komunikaty **aktualizujące**,
 - natomiast zmiany wynikające z DELETE generują komunikaty **usuwające**.
 - Odbiorca takiego strumienia powinien mieć wiedzę na temat klucza, aby w sposób poprawny interpretować i obsługiwać pojawiające się w strumieniu komunikaty.

Typy strumieni

- W Apache Flink mamy do czynienia z jeszcze jednym typem strumieni
 - **Strumień z wycofaniami (*retract stream*)** – zawiera dwa typy komunikatów/zdarzeń
 - ***dodających*** (*add messages*) i
 - ***wycofujących*** (*retract messages*).

Tabela dynamiczna jest konwertowana na strumień z wycofaniami generując

- ***dodające*** komunikaty dla każdej operacji INSERT,
- ***wycofujące*** komunikaty dla operacji DELETE
- operacja UPDATE generuje natomiast ***wycofujący*** komunikat dla poprzedniego stanu oraz ***dodający*** komunikat dla nowego stanu.

Metadane

- Tworzenie strumieni *changelog* zwykle ma miejsce w oparciu o tabele dokonujące agregacji.
- Tabele takie określając klucz (GROUP BY, PRIMARY KEY) wprowadzają dodatkowe informacje do utrzymywanej tabeli
- Znajomość tych metadanych jest kluczowa przy poprawnej interpretacji danych pojawiających się w strumieniu
 - system przetwarzania strumieni danych – kolejny krok
 - zewnętrzny odbiorca – ujście

Przetwarzanie strumieni danych w systemach Big Data

część 5 – elementy zaawansowane

Krzysztof Jankiewicz

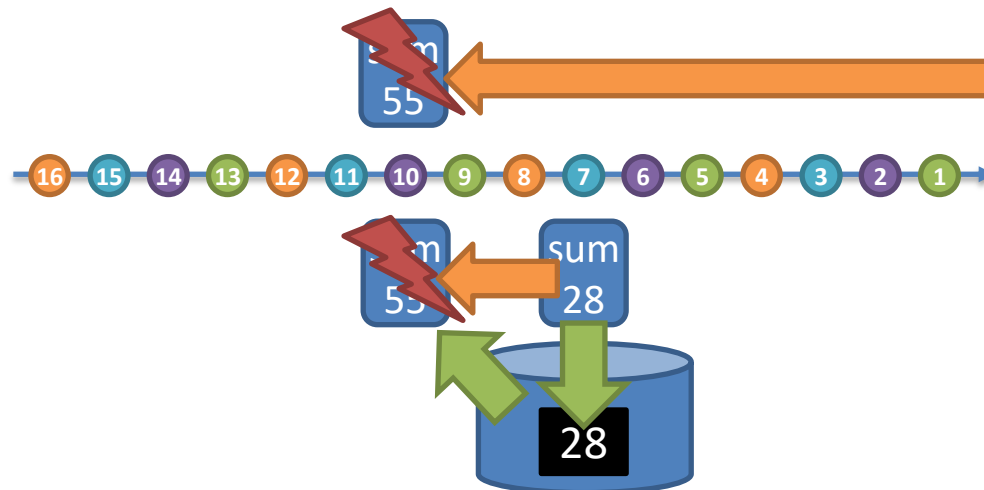
Plan

- Strumienie i ich partycje
- DAG – topologia
 - Logiczne grafy przepływu danych
 - Fizyczne grafy przepływu danych
 - Strategie przesyłania danych
- Transformacje stanowe
 - funkcje wiązane
 - TTL
- Czyściciele (evictors)
- Punkty kontrolne i ich algorytmy
- Punkty zachowania

DO UZUPEŁNIENIA

Awarie w systemach przetwarzania strumieni danych

- Aplikacje przetwarzające strumień danych działają 24/7 w związku z tym są szczególnie narażone na awarie
- Ponowne przeliczenie danych (o ile w ogóle są one dostępne!), które były przetwarzane przez bardzo długi okres czasu, **w celu odbudowania stanu** przetwarzania, może być bardzo czasochłonne i trwać godziny czy dni
- Dlatego tak bardzo istotnym jest możliwość okresowego utrwalania i zabezpieczania stanu przetwarzania, a następnie przywracania go w przypadku awarii
- Takie zabezpieczenie nosi nazwę punktów kontrolnych (na podobieństwo punktów kontrolnych w bazach danych)



Punkty kontrolne

- **Punkt kontrolny** to zapis stanu przetwarzania aplikacji na **nośnikach** zdolnych przetrwać awarie.
 - Punkty kontrolne wykorzystywane są w szczególności z dwóch powodów:
 - aby **materializować stan obliczeń** w celu uniknięcia czasochłonnego przeliczania tego stanu z danych źródłowych
 - **umożliwić ponowne uruchomienie programu sterownika**, który na podstawie zapisów w punkcie kontrolnym będzie wiedział jak odtworzyć stan przetwarzania i to przetwarzanie kontynuować.
 - Czasami rozróżnia się dwa typy punktów kontrolnych:
 - punkty kontrolne **metadanych** (*metadata checkpointing*), zawierające:
 - dane dotyczące konfiguracji – wymagane do restartu aplikacji
 - zbiór operacji definiujących aplikację przetwarzającą - DAG
 - "niedokończone" porcje danych – porcje danych, których przetwarzanie nie zostało zakończone
 - punkty kontrolne **danych** (*data checkpointing*), zawierające pośrednie etapy przetwarzania transformacji stanowych, dzięki czemu znacząco może zostać ograniczona:
 - liczba źródłowych porcji danych, oraz
 - czas i zasoby
- Punkty kontrolne danych wymagane są do odzyskania stanu przetwarzania

Punkty kontrolne, punkty zachowania i odtwarzanie stanu

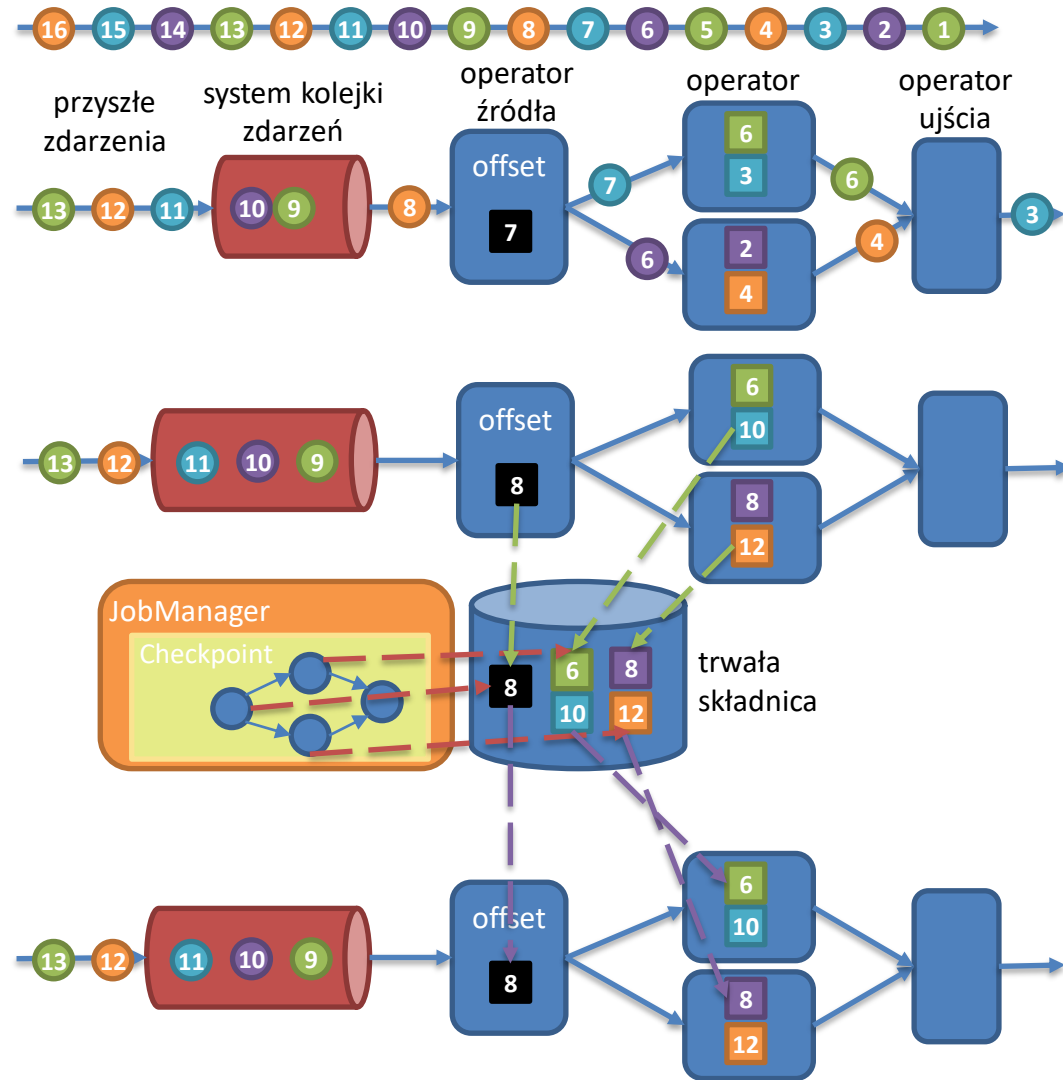
- Przetwarzanie **wsadowe** – **ponowne** pełne obliczenia
- Przetwarzanie **strumieni danych**:
 - **punkty kontrolne** (*checkpointing*) i
 - **ponowne przetwarzanie** strumienia **ostatnich** zdarzeń (*stream replay*)
- Punkty kontrolne wykonywane są **automatycznie** przez system, w celu ich ewentualnego wykorzystania podczas **odtwarzania** systemu po awarii.
- **Spójny punkt kontrolny** – to, dla systemu przetwarzania strumienia zdarzeń opartego na stanach, kopia stanu dla każdej jednostki zadania w momencie, w którym przetworzyły one określony strumień wejściowy.
- **Odtwarzanie** polega na **przywróceniu stanu** wszystkich jednostek zadań, a następnie **dostarczeniu** na wyjście wszystkich **zdarzeń**, które **nie** zostały **uwzględnione** w zapisanych stanach.
- **Punkty zachowania** – to odmiana punktu kontrolnego zapisanego wraz z dodatkowymi metadanymi wykonywana przez administratora systemu
- Celem punktów zachowania jest dla przykładu rekonfiguracja lub migracja przetwarzania

Algorytmy punktów kontrolnych

- Wariant podstawowy – z zatrzymaniem przetwarzania
- Warianty nie wymagające zatrzymania aplikacji (na przykładzie Flinka)
 - Wariant wyrównujący
 - Wariant niewyrównujący

Punkt kontrolny – wariant podstawowy

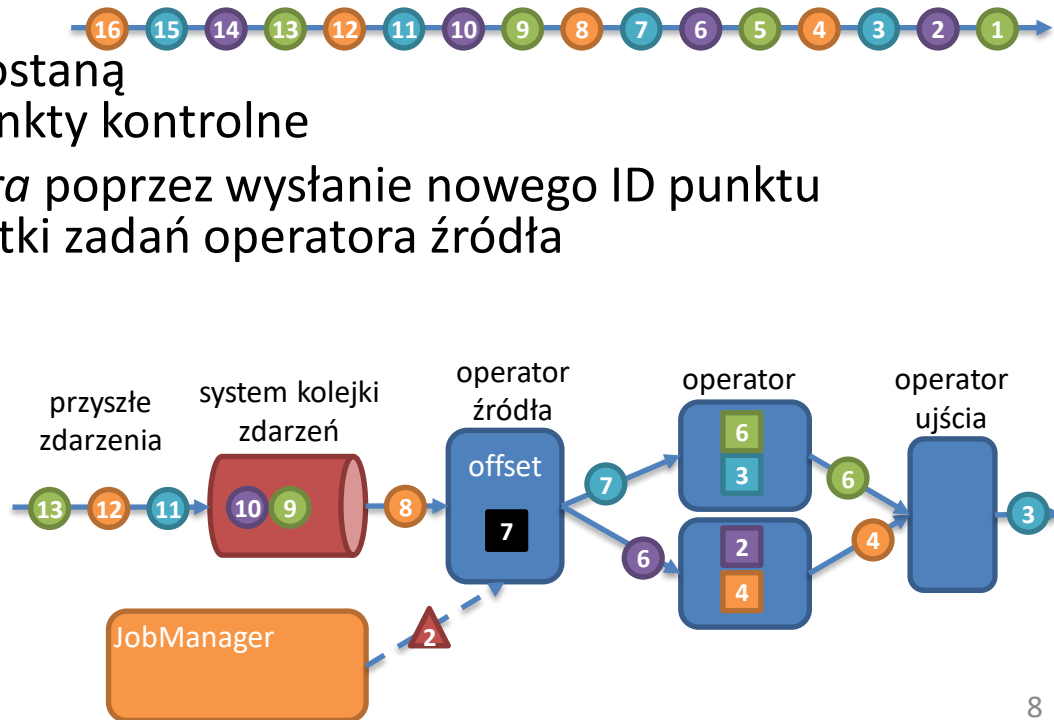
- Mógłby składać się z następujących kroków
 - Zatrzymujemy źródła i nie przyjmujemy nowych zdarzeń
 - Czekamy, aż wszystkie zdarzenia w systemie zostaną przetworzone
 - Tworzymy punkt kontrolny zapisując stany jednostek przetwarzania w trwałych repozytoriach
 - Czekamy, aż wszystkie jednostki zapiszą swój stan
 - Uruchamiamy ponownie źródła kontynuując obsługę zdarzeń
- Jakże wady takiego podejścia?
- Kroki odtwarzania
 - Przywrócenie całej aplikacji
 - Przywrócenie stanów wszystkich jednostek zgodnie z zawartością punktów kontrolnych
 - Wznowienie pracy jednostek



Jakże wady takiego rozwiązania?

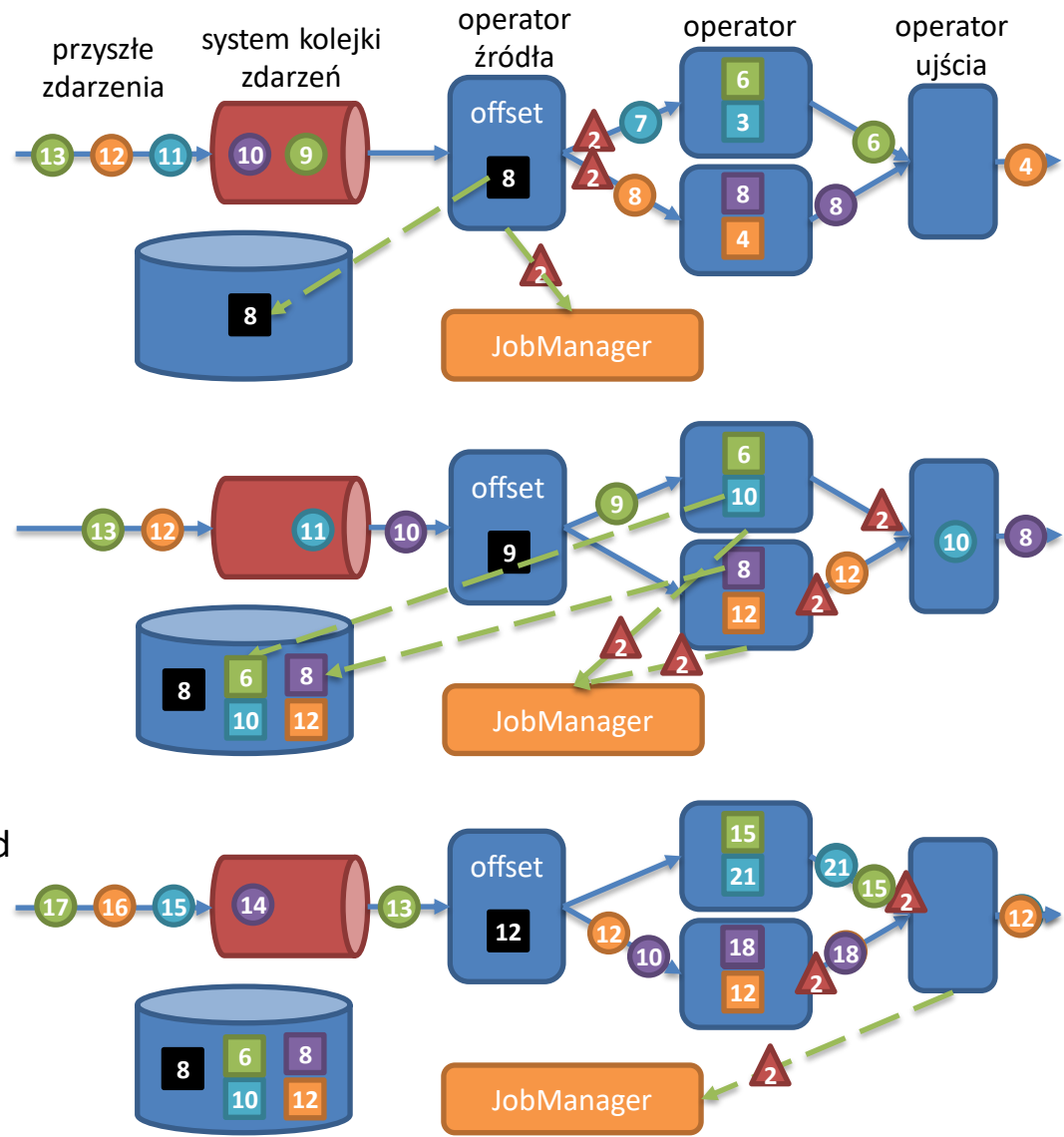
Punkt kontrolny – Flink

- Oparty jest na algorytmie Chandy–Lamport
patrz: <http://lamport.azurewebsites.net/pubs/chandy.pdf>
- Opisany szczegółowo w: <http://arxiv.org/abs/1506.08603>
- Nie wymaga zatrzymywania całej aplikacji: podczas gdy część operatorów dokonuje zapisywania swojego stanu, pozostałe dokonują standardowego przetwarzania
- Oparte są na barierach punktów kontrolnych
 - specjalne elementy strumienia zawierające ID punktów kontrolnych, których dotyczą
 - rozdzielają zdarzenia, które zostaną objęte przez poszczególne punkty kontrolne
 - inicjowane przez *JobManagera* poprzez wysłanie nowego ID punktu kontrolnego do każdej jednostki zadań operatora źródła
- Występuje w dwóch wersjach:
 - wyrównującej (*aligned*)
 - niewyrównującej, od wersji 1.11 (*unaligned*)



Punkt kontrolny wyrównujący

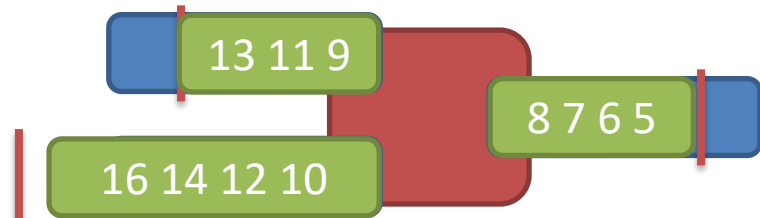
- Bariery punktu kontrolnego
 - nie wyprzedzają zdarzeń
 - są propagowane do wszystkich kolejnych operatorów
- Wyrównanie: operator, który otrzymuje barierę
 - wstrzymuje odbieranie zdarzeń od nadawcy bariery
 - odbiera zdarzenia od pozostałych nadawców i dokonuje ich przetwarzania
 - w momencie gdy otrzyma barierę od ostatniego z nadawców
 - rozpoczyna asynchroniczne zapisywanie swojego stanu
 - propaguje barierę do swoich odbiorców
 - potwierdza wykonanie punktu kontrolnego do *JobManager*
 - kontynuuje przetwarzanie zdarzeń od wszystkich nadawców
- Koniec punktu kontrolnego następuje w momencie, gdy *JobManager* otrzymuje potwierdzenia od wszystkich operatorów
- Odtwarzanie jest analogiczne do wariantu podstawowego



Jakie wady takiego rozwiązania?

Punkt kontrolny niewyrównujący

- Pojawił się w wersji 1.11, zbliżając rozwiązanie stosowane przez Flinka do algorytmu Chandy-Lamporta
- Sposób obsługi bariery:
 - operator, który otrzymuje po raz pierwszy barierę punktu kontrolnego o określonym ID rozpoczyna od razu jej przetwarzanie
 - natychmiast propaguje ją do odbiorców wyprzedzając zdarzenia zawarte
 - w buforach wejściowych, a także
 - w buforach wyjściowych operatora
 - wszystkie zdarzenia wyprzedzone przez barierę oraz te, które dojdą od pozostałych odbiorców aż do czasu otrzymania od nich tej samej bariery, są oznaczane przez operator i stają się częścią jego stanu zapisanego w składnicy
- Stan punktu kontrolnego obejmuje zatem:
 - oprócz stanu operatora także
 - zdarzenia buforów wyjściowych oraz
 - zdarzenia buforów wejściowych, które zostały wyprzedzone przez barierę
- Odtwarzanie wymaga odtworzenia wszystkich powyższych składowych operatora
- Przydatne dla aplikacji, w których propagowanie zdarzeń przez system może trwać bardzo długo. Patrz:
<https://nightlies.apache.org/flink/flink-docs-master/docs/concepts/stateful-stream-processing/#unaligned-checkpointing>



Zakres zmian aplikacji a punkty kontrolne

- Wprowadzenie
- Przykładowe akceptowane zmiany
- Zmiany, które mogą być nieakceptowane

Zakres zmian aplikacji a punkty kontrolne

- Aplikacja wznowiająca przetwarzanie na podstawie punktu kontrolnego, w założeniu jest tą samą, która go zapisała
- Możliwy jest jednak przypadek, w którym aplikacja od momentu zatrzymania/awarii uległa pewnym zmianom.
- Zakres tych zmian zależy od systemu przetwarzania strumieni danych
- Powody zmian aplikacji mogą być różne
 - poprawka z powodów biznesowych
 - poprawka ze względu na zmiany środowiska po awarii

Przykładowe akceptowane zmiany

- Zmiany w parametrach źródeł nie zmieniające semantyki
- Zmiany typu ujęcia (niektóre) w szczególności definiowanych przez użytkownika
- Zmiany w parametrach ujęć
- Dodanie, usunięcie operatorów `filter`
- Zmiana w projekcji o ile wynikowy schemat jest identyczny
- Zmiany w funkcjach użytkownika obsługujących przetwarzanie stanowe

Przykładowe zmiany nieakceptowane

- Zmiany typów lub liczby źródeł (np. zmiana tematu Kafki)
- Zmiany w liczbie lub typach kluczy grupujących lub funkcji agregujących
- Zmiany w sposobie eliminacji zduplikowanych wartości
- Zmiany w definicjach operacji połączenia strumieni danych
- Zmiany w postaci stanów oraz sposobie obsługi ich terminów ważności dla funkcji stanowych użytkownika