Przetwarzanie strumieni danych na przykładzie środowiska Esper

podstawy

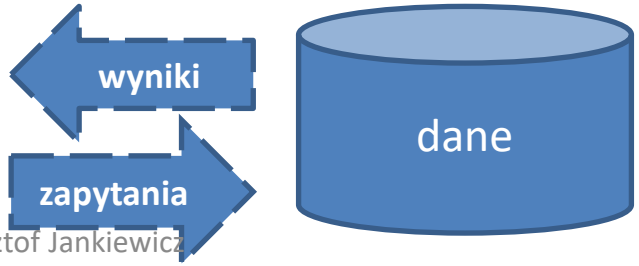
Krzysztof Jankiewicz
Marek Wojciechowski

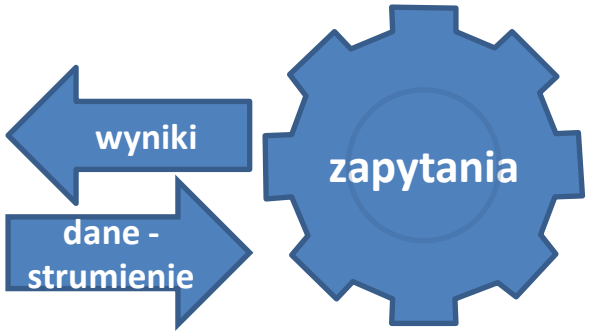
Wprowadzenie

- Wsadowe przetwarzanie danych a przetwarzanie strumieni danych
- Architektura systemu przetwarzania strumieni danych
- Podstawowe definicje
- Typy zdarzeń
- Relacja, okno, strumienie wynikowe relacji
- Tworzenie strumienia zdarzeń
- Rejestrowanie zapytań
- Przechwytywanie strumieni wynikowych relacji

Przetwarzanie wsadowe danych vs przetw. strumieni (DBMS vs. DSMS)

Wsadowe (DBMS)	Przetwarzanie strumieni (DSMS)
dane trwałe	dane ulotne, możliwość skorzystania z danych trwałych
ograniczone zbiory danych	nieograniczone strumienie danych
nieograniczona przestrzeń dyskowa	ograniczona pamięć operacyjna
zapytania uruchamiane ad hoc lub wsadowo	zapytania przetwarzane w sposób ciągły (*standing queries*)
zapytania są częścią zewnętrzną systemu	zapytania stanowią część integralną systemu
możliwość stosowania algorytmów wieloprzebiegowych	konieczność stosowania algorytmów jednoprzebiegowych (*you only get one look*)
plan realizacji zapytania opracowany na określonym rozkładzie danych	plan realizacji zapytania opracowany na zmiennym i nieznanym rozkładzie danych





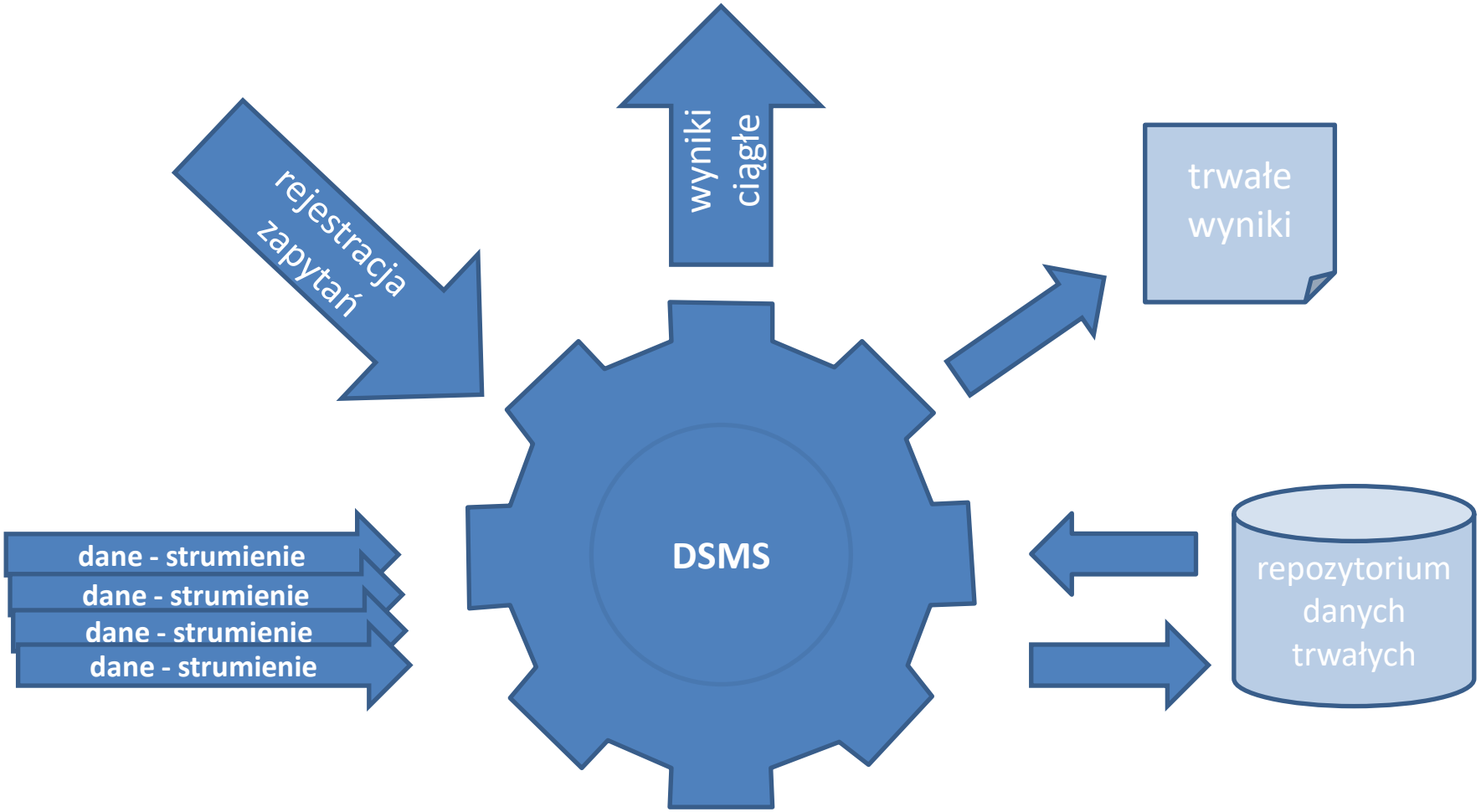
Obszary zastosowań

- Zarządzanie procesami biznesowymi i ich automatyzacja
  - monitorowanie procesów
  - oprogramowanie klasy BAM (business activity monitoring)
  - raportowanie wyjątków
- Rynki finansowe
  - algorithmic trading – automatyzowanie działań na rynkach finansowych,
  - wykrywanie oszustw
  - zarządzaniem ryzykiem
- Monitorowanie sieci i aplikacji
  - wykrywanie włamań
  - monitorowanie jakości dostarczanych usług
- Aplikacje oparte o sieci sensorów (Internet przedmiotów)
  - RFID
  - planowanie i kontrola linii produkcyjnych
  - kontrola lotów

Kto i jak wykorzystuje Esper?

- Firmy wykorzystujące Esper do własnych analiz
  - "We have 9 companies in the S&P 100 among our regular customers"
- Firmy integrujące Esper w ramach swoich produktów (OEM)
  - "We have 6 out of the top 25 largest software companies in the world as our OEM customers"
- Źródło: `http://www.espertech.com/customers/`

Architektura DSMS
(*Data Stream Management System*)



Esper

- Oprogramowanie klasy EDA -> ESP -> CEP
  - EDA = Event-Driven Architecture
  - ESP = Event Stream Processing
  - CEP = Complex Event Processing
- Rozwijane przez EsperTech Inc.
  - `http://www.espertech.com`
- Java + port na .NET (NEsper)
- Zapytania deklaratywne w języku EPL
  - rozszerzenie SQL-92
  - implementacja StreamSQL
- Esper a narzędzia Big Data (Kafka, Flink)?

Architektura Espera

- Esper obecnie (8.x) składa się z trzech komponentów
  - Języka EPL (*Event Processing Language)*
  - Kompilatora języka (*compiler*)
  - Środowiska uruchomieniowego (*runtime*)
- Językiem Espera jest EPL (Event Processing Language)
  - Kompilator Espera kompiluje EPL do bajtkodu JVM
  - Wynikowy kod działa na JVM w ramach środowiska uruchomieniowego Espera
  - Analogia do Kotlina, Scali, itd., ale EPL nie jest językiem imperatywnym

Co zmieniło się w wersji 8.X?

- Esper w wersji 8.X zmienił się znacznie w stosunku do poprzednich wersji, ale także w wielu aspektach
  - Nowy silnik zdarzeń, który jest znacznie bardziej wydajny i skalowalny
  - Nowy język SQL, który jest bardziej zgodny z SQL
  - Nowe funkcje analityczne, funkcje agregacji, grupowanie i okna czasowe.
  - Nowe API, które jest bardziej intuicyjne i łatwiejszy w użyciu.
  - Znacznie więcej możliwości integracji z innymi systemami, np. Big Data, Streaming czy platformy Machine Learning.
- Jedna zmiana wymaga zaakcentowania
  - W wersjach wcześniejszych polecenia EPL były interpretowane przez silnik zdarzeń a następnie wykonywane.
  - Od wersji 8.0 polecenia EPL są kompilowane do kodu maszynowego przed ich wykonaniem. Oznacza to, że polecenia EPL są przetwarzane przed ich uruchomieniem, co pozwala na lepsze wykorzystanie możliwości systemu i daje lepszą wydajność.

Podstawowe definicje

- Zdarzenie – czasami zwane też krotką
  - atomowa składowa strumienia
  - zawierająca informacje – dane
  - pojawiająca się w określonym momencie – posiadająca swój znacznik czasowy
- Strumień – sekwencja zdarzeń
- Przykłady…

Strumień – Seria krotek uporządkowanych za pomocą znaczników czasowych,
Para (s; τ), gdzie: s jest krotką, a τ znacznikiem czasowym

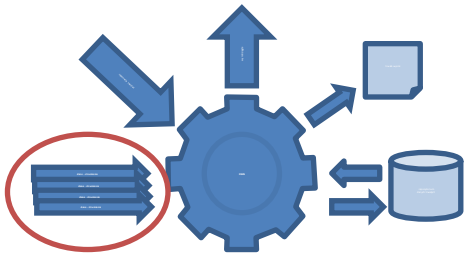
Typy zdarzeń w Esper

Klasa	opis
`java.lang.Object`	Dowolna klasa POJO
`java.util.Map`	Pary atrybutów klucz-wartość
`Object[] (array of object)`	Tablica, której każdy element jest właściwością
`String`	Zdarzenie jako dokument JSON
`org.w3c.dom.Node`	Dokument XML (DOM)
`org.apache.avro.generic.GenericData.Record`	Zdarzenia Apache Avro (system serializacji)

Atrybuty – własności zdarzeń

Typ	Opis	Przykład
Prosty	Prosty atrybut posiadający pojedynczą wartość	`kursOtwarcia`
Indeksowany	Własność będąca indeksowaną kolekcją wartości	`listaTowarow[i]`
Mapowany	Własność będąca kolekcją wartości dostępną za pomocą wartości klucza	`listaTowarow['półka']`
Zagnieżdżony	Własność zagnieżdżona	`temperatura.jednostka`

Tworzenie strumienia zdarzeń



- Klasa POJO zdarzenia

```
public class KursAkcji {
      private String spolka;
      private String market;
      private Date data;
      private Float kursOtwarcia;
      private Float wartoscMax;
      private Float wartoscMin;
      private Float kursZamkniecia;
      private Float obrot;
. . .
}
```

Konfiguracja dla kompilatora
i środowiska uruchomieniowego

```
Configuration configuration
              = new Configuration();
configuration.getCommon()
             .addEventType(KursAkcji.class);
```

Domyślną nazwą typu zdarzenia
jest prosta nazwa klasy (bez pakietu)

Uzyskanie obiektu Runtime

```
EPRuntime runtime = EPRuntimeProvider.getDefaultRuntime(configuration);
```

Dostęp do interfejsu do wysyłania zdarzeń do środowiska uruchomieniowego

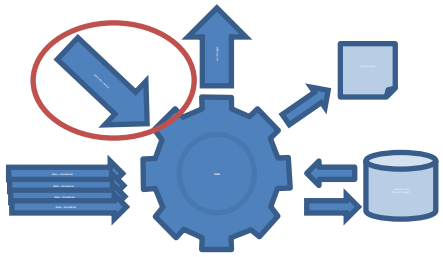
```
EPEventService eventService = runtime.getEventService();
```

Utworzenie i wysłanie zdarzenia

```
kurs = new KursAkcji(. . .);

eventService.sendEventBean(kurs, "KursAkcji");
```

Kompilacja zapytań



Uzyskanie obiektu kompilatora

```
EPCompiler compiler = EPCompilerProvider.getCompiler();
```

Dostarczenie konfiguracji jako argumentów kompilatora

```
CompilerArguments args = new CompilerArguments(configuration);
```

Źródłowe zapytanie EPL

```
String epl = "select irstream spolka as X, kursOtwarcia as Y " +
             "from KursAkcji#length(5)"
```

Nadanie nazwy zapytaniu EPL

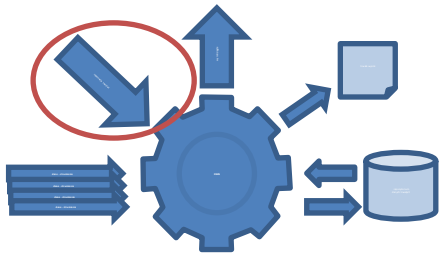
```
String epl = "@name('myStmtName') select irstream spolka as X, kursOtwarcia as Y " +
             "from KursAkcji#length(5)"
```

Kompilacja polecenia EPL

```
EPCompiled epCompiled;

try {
  epCompiled = compiler.compile(epl, args);
}
catch (EPCompileException ex) {
  // handle exception
  throw new RuntimeException(ex);
}
```

Rejestracja zapytań



Uzyskanie obiektu Runtime

```
EPRuntime runtime = EPRuntimeProvider.getDefaultRuntime(configuration);
```

Dostęp usługi rejestracji zapytań

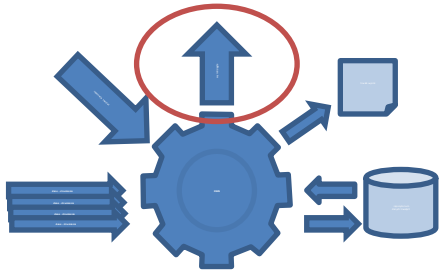
```
EPDeploymentService deploymentService = epRuntime.getDeploymentService();
```

Rejestracja (*deployment*) zapytania

```
EPDeployment deployment;

try {
  deployment = deploymentService.deploy(epCompiled);
}
catch (EPDeployException ex) {
  // handle exception
  throw new RuntimeException(ex);
}
```

Pobieranie wyników relacji



- Podłączanie się do wyników zapytań możliwe jest za pomocą:
  - obiektu implementującego interfejs `UpdateListener`
    - Obiekty te otrzymują instancje obiektów `EventBean` zawierające wyniki zapytań
    - Silnik dostarcza wyniki zapytań do wszystkich listenerów, którzy dla nadanego zapytania zostali zarejestrowani
    - Do rejestrowania listenera służy metoda `addListener` (wyrejestrowania `removeListener`) wywoływana na rzecz zapytania (`EPStatement`)
  - obiektu Subscriber
    - Klasa POJO
    - Wyspecyfikowana metoda, której nagłówek odpowiada postaci zdarzenia, odbiera rezultaty zapytania
    - Najszybszy ze sposobów
    - Tylko jeden Subscriber
    - Do rejestracji służy metoda `setSubscriber` obiektu zapytania
  - Pull API
    - Wykorzystując `safeIterator` i `iterator` aplikacja zgłasza się po wyniki z danego zapytania
    - Wyniki obierane są jako `java.util.Iterator<EventBean>`
    - Wyniki nie są odbierane w sposób ciągły

Podłączanie obiektu implementującego interfejs UpdateListener do wyników zapytania

```
public class ProstyListener implements UpdateListener {
  @Override
  public void update(EventBean[] newEvents, EventBean[] oldEvents) {
    if (newEvents != null) {
      for (int i = 0; i < newEvents.length; i++) {
        System.out.println("ISTREAM : " + newEvents[i].getUnderlying());
      }
    }
    if (oldEvents != null) {
      for (int i = 0; i < oldEvents.length; i++) {
        System.out.println("RSTREAM : " + oldEvents[i].getUnderlying());
      }
    }
  }
}
```

Klasa listenera

Uzyskanie obiektu polecenia

```
EPDeploymentService deploymentService = runtime.getDeploymentService();
```

```
EPDeployment deployment = deploymentService.deploy(epCompiled);
```

```
EPStatement[] statements = deployment.getStatements()
```

```
EPStatement statement = deploymentService
                        .getStatement(deployment.getDeploymentId(), "myStmtName")
```

Podłączenie listenera

```
ProstyListener listener = new ProstyListener();
statement.addListener(listener);
```

Zdarzenia w formacie JSON

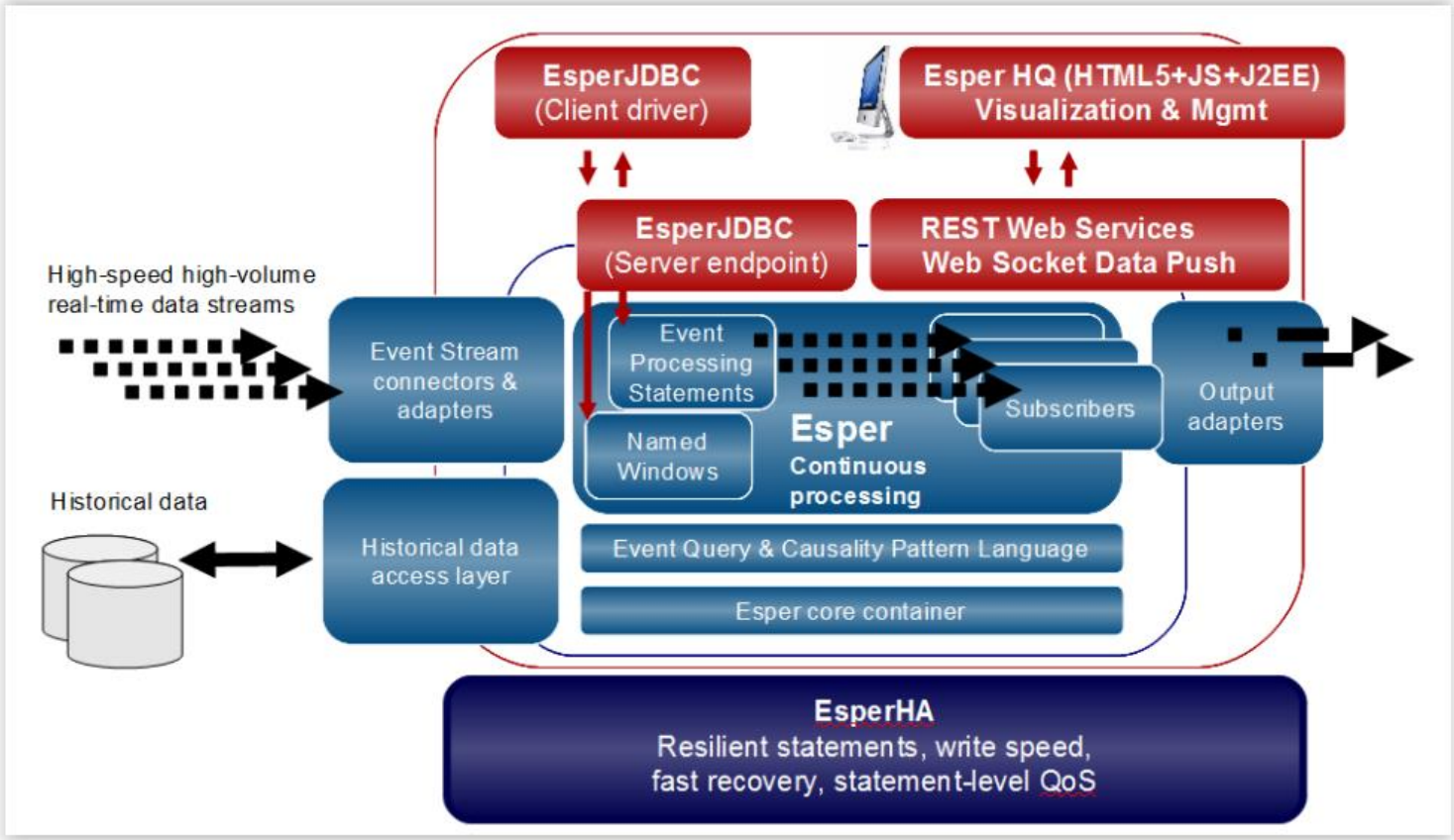
- Nie trzeba tworzyć klasy ani jej rejestrować
- Rejestracja typu zdarzenia odbywa się za pomocą polecenia EPL

```
epCompiled = compiler.compile("""
  @public @buseventtype create json schema
     Punkty(house string, character string, score int, ts string);
  @name('my-statement') select * from Punkty#length(5);""",  compilerArgs);
```

Lambda zamiast obiektu Listenera

```
statement.addListener( (newData, oldData, stmt, runTime) -> {
    for (EventBean eventBean : newData) {
        String house = (String) eventBean.get("house");
        String character = (String) eventBean.get("character");
        int score = (int) eventBean.get("score");
        System.out.println(String.format("house: %s, character: %s, score: %d",
                                         house, character, score));
}
```

EsperHA



- Skalowalność (korzystająca z Zeppelin, Kafka i Kafka Streams)
- GUI, REST web services, JDBC
- Trwałość stanu (zapytań, zdarzeń)
- Złącza pozwalające korzystać ze zdalnych repozytoriów (jako źródeł lub jako ujść)
- Obsługa punktów kontrolnych, pozwalająca na zapisywanie stanu przetwarzania i odtwarzanie

http://static.espertech.com/EsperTech%20technical%20datasheet.pdf

Relacja, a strumienie

- Relacja – zmienny w czasie zbiór krotek
- Relacja definiowana jest w oparciu o zapytanie oparte na zmiennym w czasie strumieniu zdarzeń wyjściowych
- W efekcie zmienności strumienia wejściowego zawartość relacji również ulega zmianie:
  - pojawiają się nowe zdarzenia (krotki)
  - znikają inne zdarzenia (krotki)
- Prowadzi to do tworzenia przez relacje dwóch strumieni wynikowych
  - strumienia zdarzeń (krotek) usuwanych
  - strumienia zdarzeń (krotek) wstawianych
- Postać zdarzeń w strumieniach wynikowych może różnić się od postaci zdarzeń strumienia wejściowego

Okna

- Zapytania definiujące relacje działają z reguły na jego najnowszym fragmencie (ostatniej serii zdarzeń)
- Dlaczego?
- Zakres zdarzeń i zmienność tego zakresu definiowana jest za pomocą okna (podobnego do okien znanych z funkcji analitycznych SQL)
- Podstawowe typy okien
  - czasowe
  - czasowe wsadowe (batch)
  - długościowe
  - długościowe wsadowe

Składnia EPL dla okien

- Okna w języku EPL mają przypisaną nazwę (*name*) i przestrzeń nazw (*namespace*)
- Przestrzenie nazw okien w EPL: `win`, `std`, `ext`, `stat`
- W aktualnej wersji języka EPL nie jest konieczne jawne poprzedzanie nazwy okna przestrzenią nazw
- Trzy warianty składni okien (na przykładzie `win:length`):

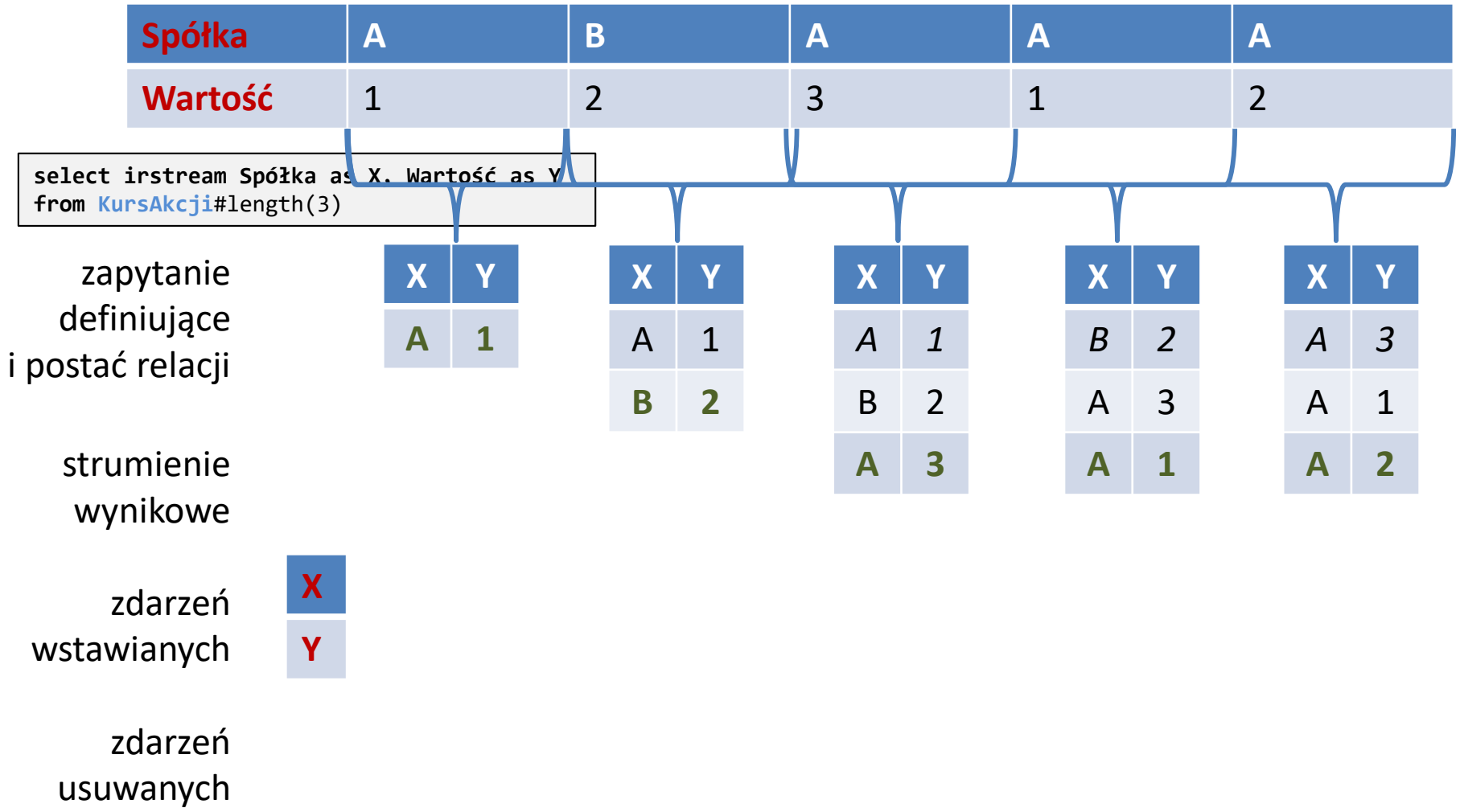
```
select irstream Spółka as X, Wartość as Y
from KursAkcji.win:length(3)
```

```
select irstream Spółka as X, Wartość as Y
from KursAkcji#win:length(3)
```

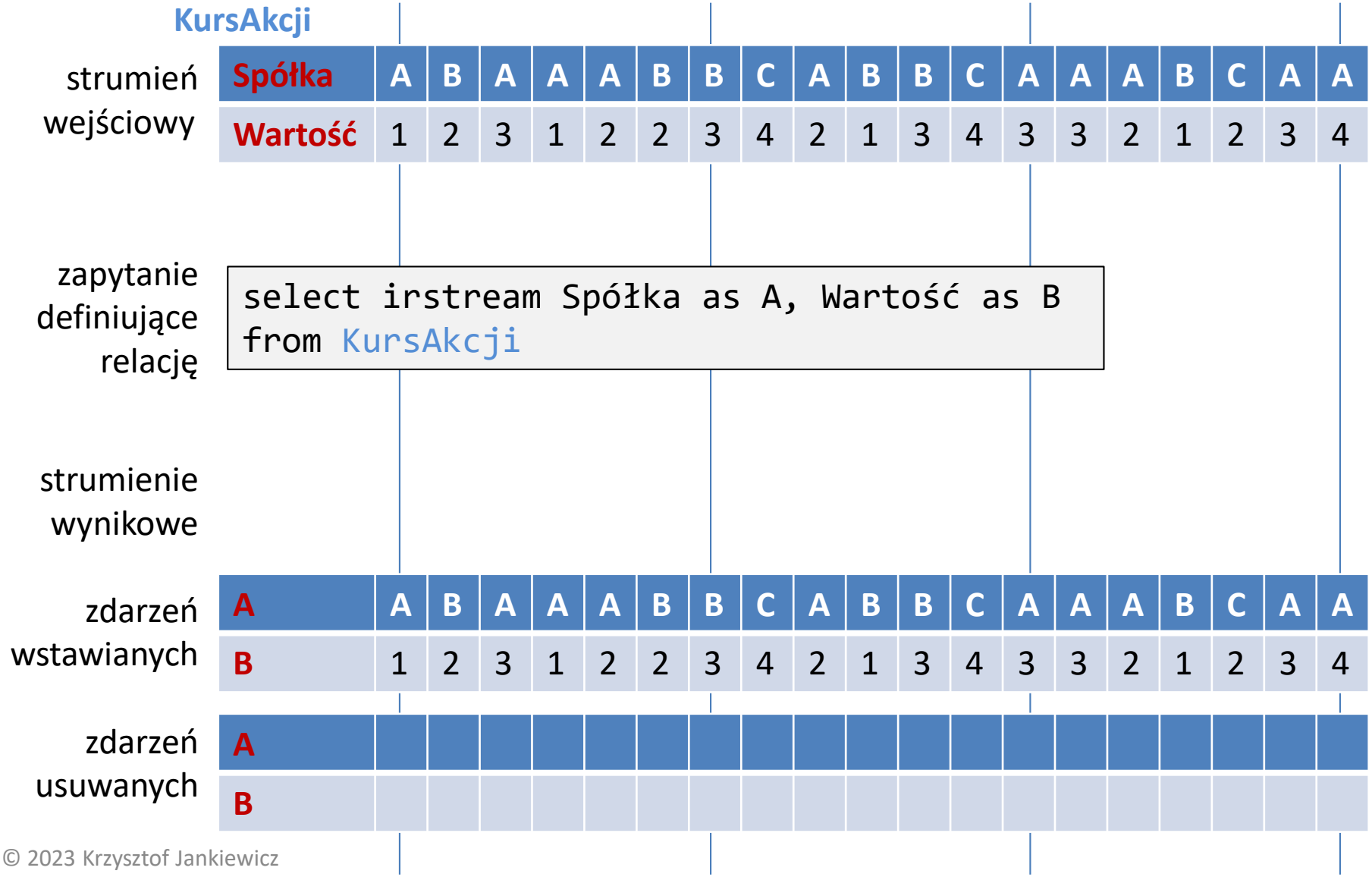
```
select irstream Spółka as X, Wartość as Y
from KursAkcji#length(3)
```

Aktualnie preferowana wg dokumentacji (8.x)

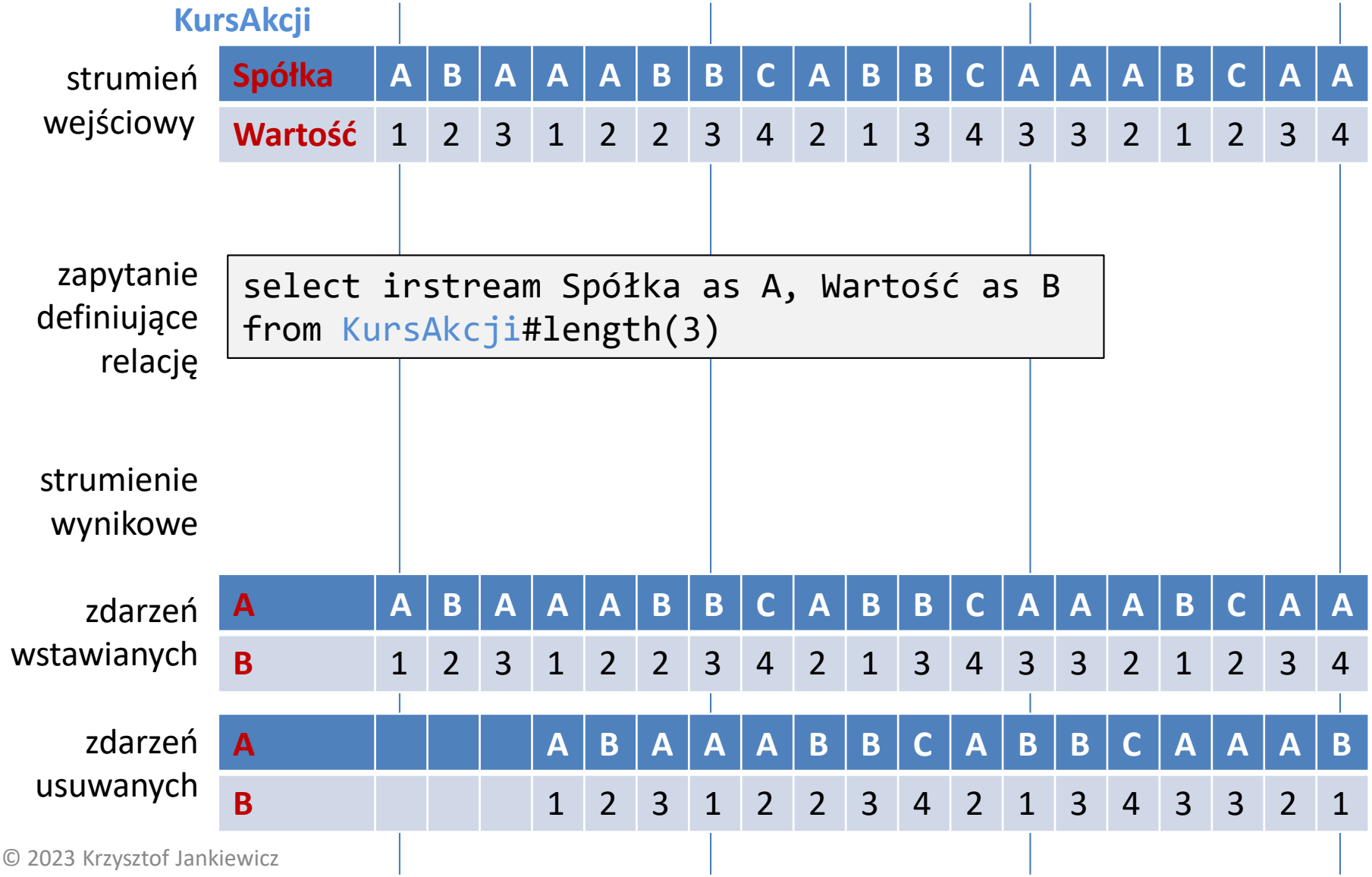
Zmienność relacji a strumienie wynikowe



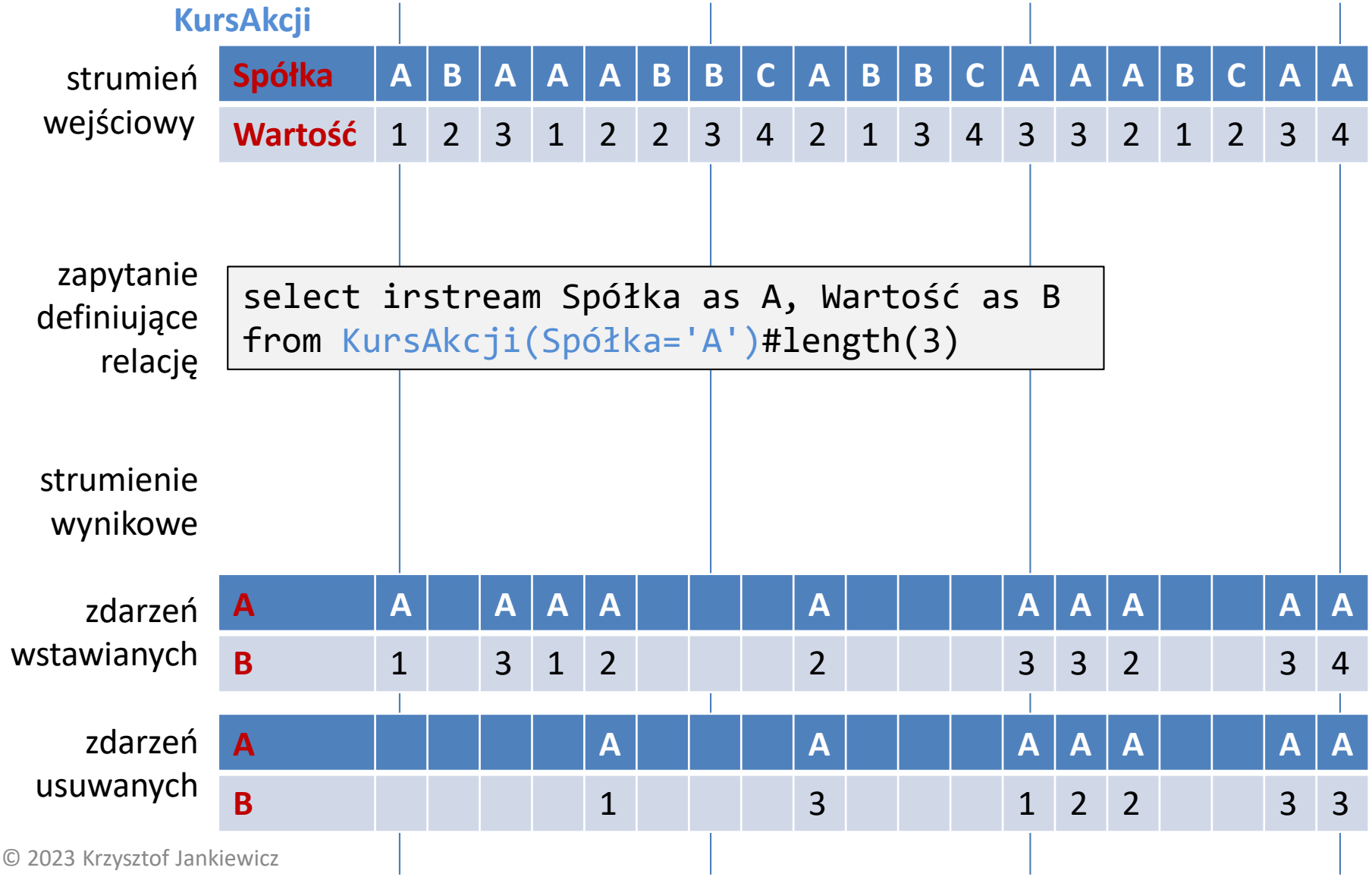
Przykład 1



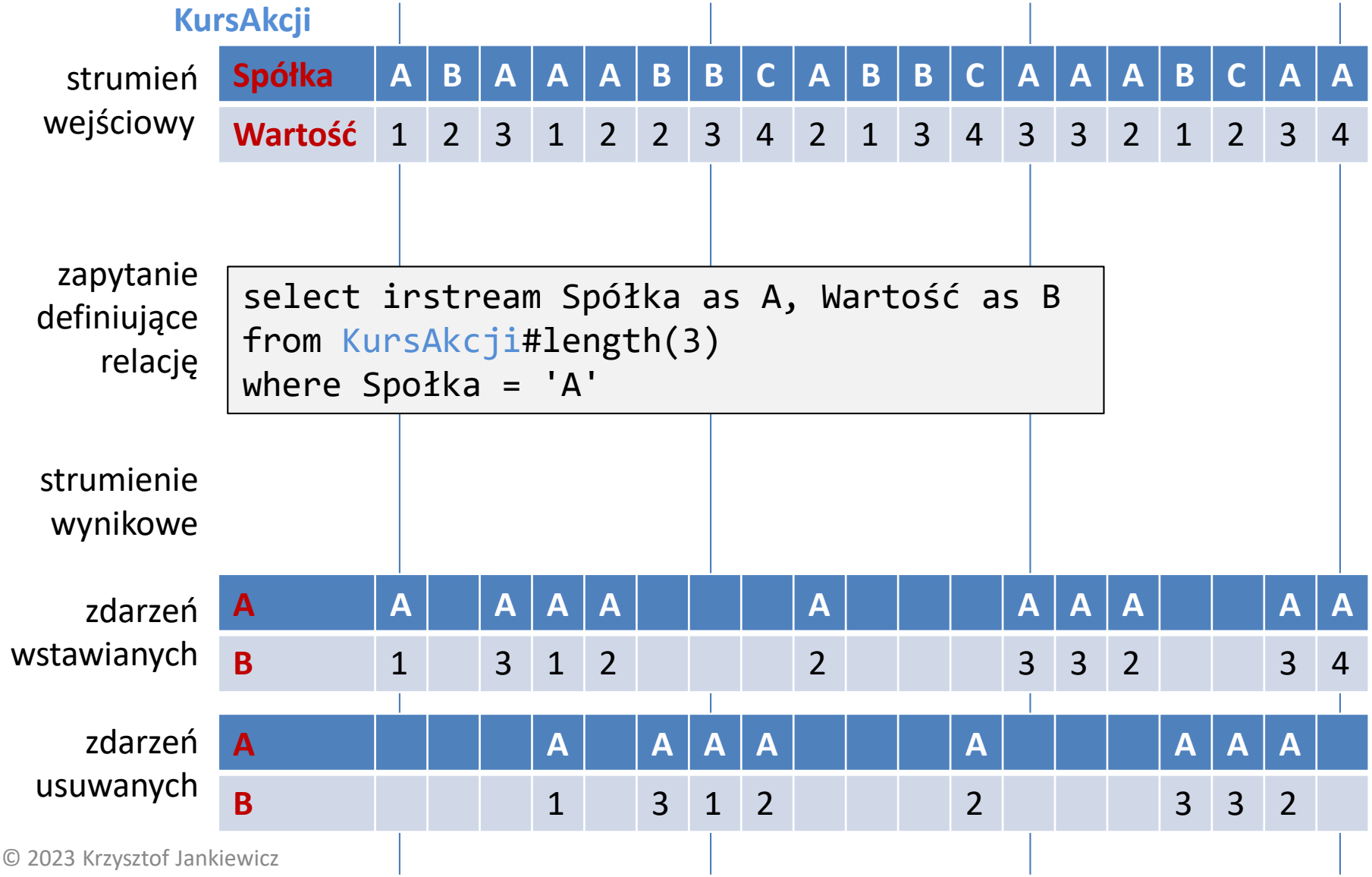
Przykład 2



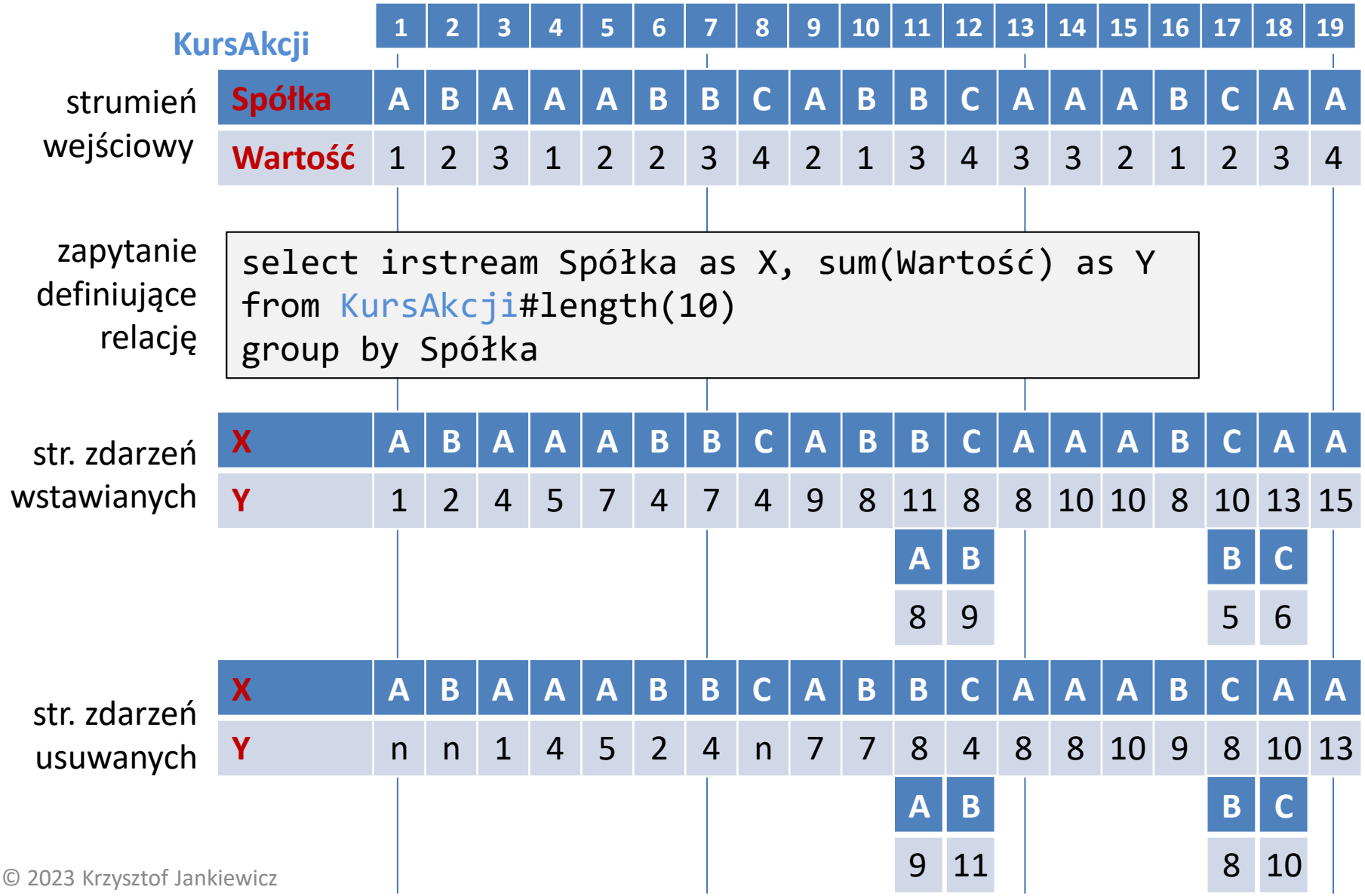
Przykład 3



Przykład 4



Przykład 5



Podsumowanie

- Klasyczne a strumieniowe przetwarzanie danych
- Architektura systemu przetwarzania danych strumieniowych
- Podstawowe definicje
- Typy zdarzeń
- Relacja, okno, strumienie wynikowe relacji
- Tworzenie strumienia zdarzeń
- Rejestrowanie zapytań
- Przechwytywanie strumieni wynikowych relacji