

Złączenia w modelu Map/Reduce

Maciej Penar

Autor

Maciej Penar



- Typ Szalonego Naukowca - „Udowodnię Ci że to zadziała (nie powinno)”
- Aktualnie związany z firmą Vulcan S.p. z o.o.
- Prywatnie pesymista i sceptyk





Agenda

- Model programistyczny
 - Idea
- Złączenia
 - Podejście klasyczne
 - Podejście z równoważeniem obciążenia
 - Hive
 - Wydajność
- Podsumowanie



Map-Reduce

Rok 2004 (grudzień)

Google publikuje artykuł:

„MapReduce: Simplified Data Processing on Large Clusters”,

J.Dean & S.Ghemawat,

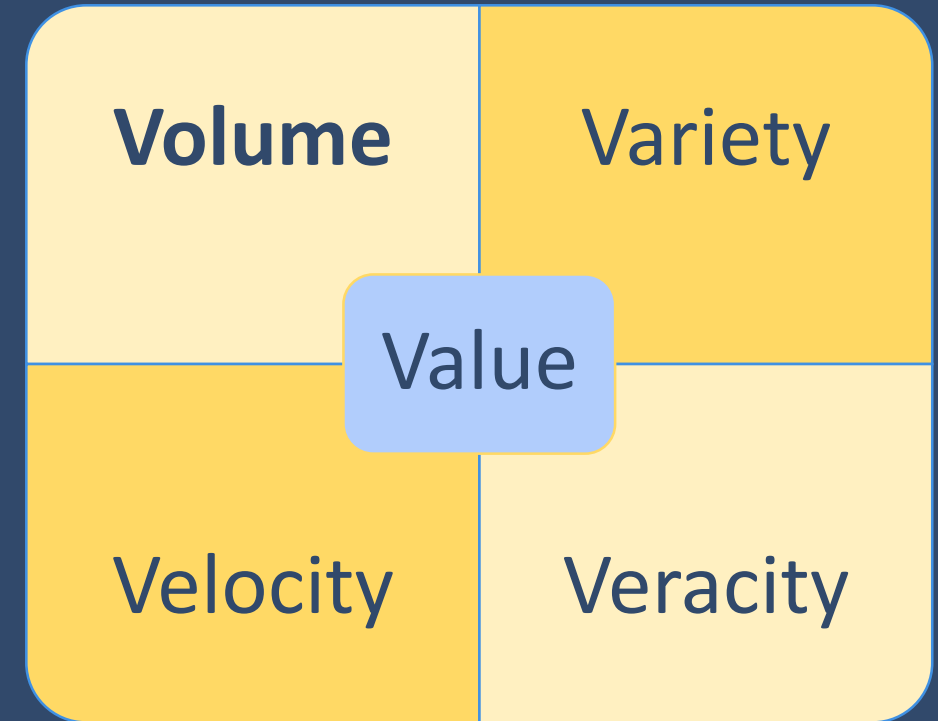
Sixth Symposium on Operating System Design and Implementation*

* Link: <http://static.googleusercontent.com/media/research.google.com/pl//archive/mapreduce-osdi04.pdf>

Map-Reduce

Wykorzystanie

- Najczęściej we Frameworkach/Systemach Map/Reduce np.:
 - Google Map/Reduce
 - Hadoop
 - Stratosphere (model PACT)
- Systemy najczęściej wiązane z trendem Big Data
- Źródła w Java
- Duże wolumeny
- Czasochłonne zapytania
- Optymalizacja pod skan



Map-Reduce

Cechy ogólne

- System rozproszony, dwa typy węzłów:
 - MasterNode (*NameNode) – odpowiedzialny za indeksowanie plików
 - ChunkServer (*WorkerNode) – odpowiedzialny za działanie agentów
- Dwie warstwy działające niezależnie, realizowane agentami:
 - **Obliczeniowa**
 - Przechowująca dane (w postaci surowej)
- System Google'a jest do użytku wewnętrznego
- Model programistyczny Map/Reduce

Map-Reduce

Idea

Model projektowania programów o dużym stopniu równoleglenia.

Program Map/Reduce składa się z dwóch faz:

1. Mapowania/Grupowania – odpowiadają za nią tzw. Mappery, wywołujące 1 raz dla każdej cząstki danych metodę $\text{map}(k, v)$ – wartości przekazywane najczęściej jako v
2. Redukcji/Agregacji – odpowiadają za nią tzw. Reducery, wywołujące 1 raz dla każdej grupy o kluczu k metodę $\text{reduce}(k, \text{list}\langle v \rangle)$

Dodatkowo istnieje metoda $\text{write}(k, v)$ wiążąca wartość v z kluczem k

Mappery i Reducery mają cykl życia `setup-map/reduce-cleanup`

Hadoop

Otwarta implementacja Map Reduce



Model programistyczny

Idea

Czy to wszystko?

... w rozważaniach teoretycznych przyjmuje się następujący model:

```
map(k1,v1) -> list(k2,v2)  
reduce(k2, list(v2)) -> list(v3)
```

Ze względu na prostotę D.J. DeWitt i M.Stonebreaker opublikowali artykuł zatytułowany: **MapReduce: A major step backwards** [link](#)

Wracając do pytania: nie do końca... ale wrócimy do tego

Model programistyczny

Przykład

Suchy przykład: zliczanie słów

Założmy, że mamy pewien zbiór plików/plik.

Chcemy zliczyć częstość występowania każdego słowa.

Plik.txt

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Lorem ipsum dolor sit amet. Tristique ex quis. Luctus et
ultrices posuere cubilia Curae.
Sed maximus molestie eros, id cursus libero
condimentum eget.
Fusce in leo congue, tristique ex quis, egestas diam.
Etiam et porttitor lorem.
Mauris sit amet nisl quis tellus convallis facilisis in ut
libero.
Vestibulum ante ipsum primis in faucibus orci luctus et
ultrices posuere cubilia Curae ...

Model programistyczny

Przykład: Faza Map

Mapper

```
class WordMapper extends Mapper<Long,String,String,Integer> {  
  
    @Override  
    public void map(Long key, String value, Context context){  
        for(String word : value.split(" "))  
            context.write(word.replaceAll("\\.|,","").trim(), 1);  
    }  
}
```

// Wywołanie na poziomie pojedynczej linii
// Podzielenie wyrazów ze względu na spacje
// Każdy wyraz emituje parę <wyraz, 1>

* Tu przyda się dygresja odnośnie typów Java vs. Hadoop

Model programistyczny

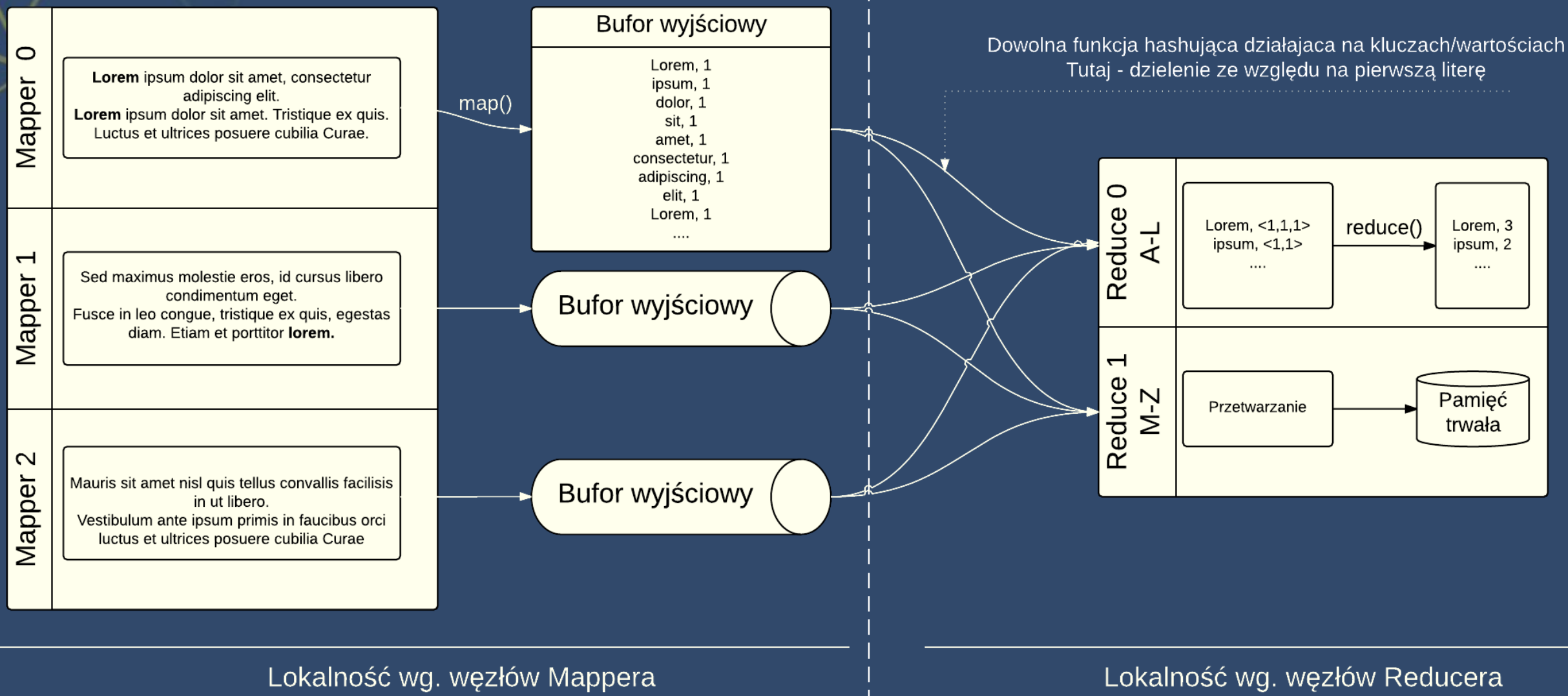
Przykład: Faza Reduce

Reducer

```
class WordReducer extends Reducer<String, Integer, String, String> {  
    @Override  
    protected void reduce(String key, Iterable<Integer> values, Context context){ // wywołanie per grupa  
        int count = 0; // inicjalizacja licznika  
        for(Integer value : values) // iteracja po wszystkich ,1'  
            ++count ; // zliczenie ich  
        context.write(key, count + ""); // zapis  
    }  
}
```

Faza Redukcji jest ostatnia, więc emisja rekordu następuje bezpośrednio do pliku

Faza SHUFFLE (przesłanie danych przez sieć)



Model programistyczny

Przykład: Combine

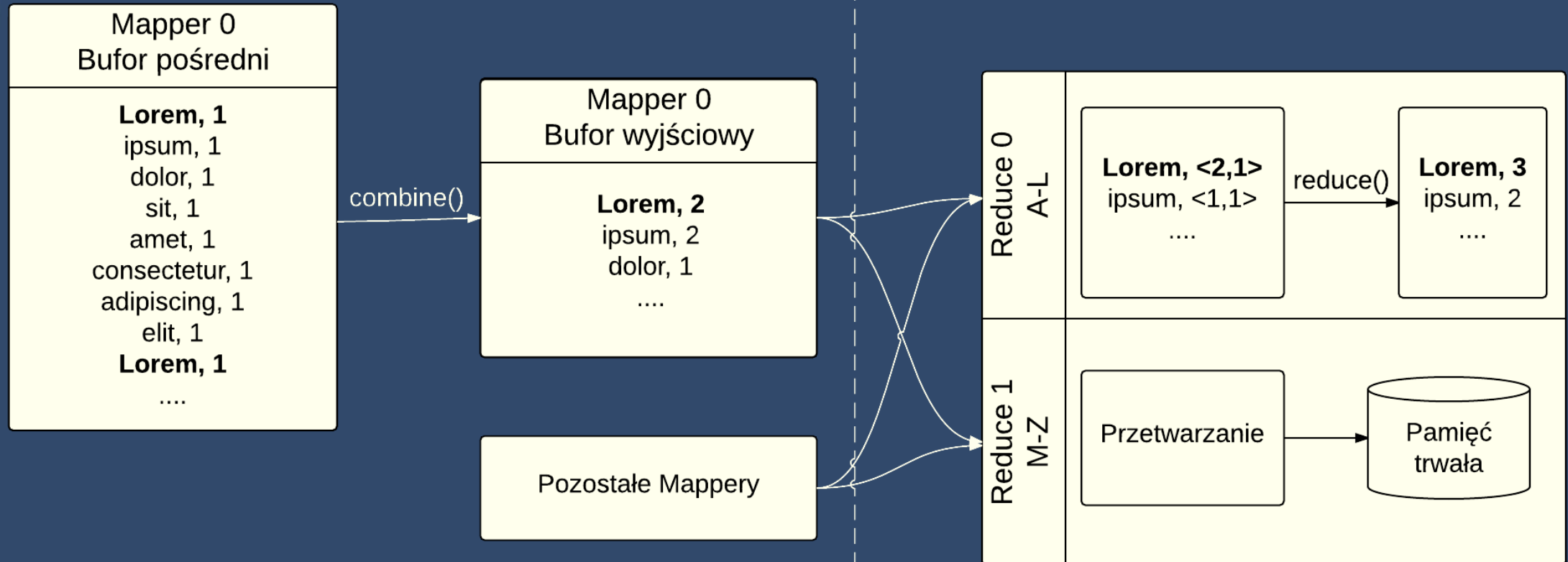
Reducer2

```
class WordReducer2 extends Reducer<String, Integer, String, String> {  
    @Override  
    protected void reduce(String key, Iterable<Integer> values, Context context){ // wywołanie per grupa  
        int count = 0; // inicjalizacja licznika  
        for(Integer value : values) // iteracja po wszystkich ,1'  
            count += value; // zliczenie liczników  
        context.write(key, count + ""); // zapis  
    }  
}
```

Podczas konfiguracji zadania (Job/Tool) możemy odpowiedzieć Hadoopowi, żeby wykonał:

- Lokalną Redukcję (na buforze Mapper'a) – za pomocą klasy WordReducer
- Fazę Redukcji – za pomocą klasy WordReducer2

Faza SHUFFLE
(przesłanie danych przez sieć)



Lokalność wg. węzłów Mapperów

Lokalność wg. węzłów
Reducerów

Model programistyczny

Czy to wszystko? / Zaczynij te złączenia

Co powinniśmy wiedzieć:

- Konfiguracja – niestety pominąłem, to nie wykład o programowaniu w MR
- Faza Redukcji zaczyna się tylko wtedy gdy faza Mapowania się zakończyła
- Determinizm (zazwyczaj)
- Dane na Reducer'ach są posortowane wg. porządku klasy klucza (Comparable*)
- **Wywołanie reduce() jest na poziomie wykrycia zmiany wartości zwracanej tzw. komparatora grupującego – GroupingComparator (domyślnie wywołania compareTo() na kluczach) -> najczęściej: sortuje po <A,B>, grupuje <A>**
- Partitioner - klasa dzieląca dane na Reducer'y (fizyczna separacja): np. dzielę dane ze względu na <A>, sortuje po <B, C>, grupuje po - szczyt finezji

* WritableComparable, link: <https://hadoop.apache.org/docs/r2.7.0/api/org/apache/hadoop/io/WritableComparable.html>

Model programistyczny

Mój stosunek do modelu Map Reduce

1. „W rozważaniach teoretycznych przyjmuje się następujący model”

```
map(k1,v1) -> list(k2,v2)  
reduce(k2, list(v2)) -> list(v3)
```

2. ?

Złączenia!

Złączenia

Problem

Motywacja:

- Dane w postaci znormalizowanej
- Istnienie naturalnych zależności pomiędzy danymi

Problem:

- Złączenie N relacji (n -ta relacja oznaczana R_n) rozumiemy jako podzbiór iloczynu kartezyjskiego pomiędzy tymi źródłami, gdzie każda krotka tego podzbioru spełnia pewien warunek θ
- Za pomocą algebry relacji: $R_1 \bowtie_{\theta_1} \dots \bowtie_{\theta_{n-1}} R_n \equiv \sigma_{\theta}(R_1 \times \dots \times R_n)$
- Dla nierozproszonych BD możemy wyróżnić trzy klasy algorytmów:
 - Oparte na pętli zagnieżdżonej
 - Oparte o sortowanie (Sort-Merge join)
 - Oparte o hashowanie (Hash join)

Złączenia

Problem

SELECT

*

FROM

R_1

INNER JOIN R_2 ON θ_1

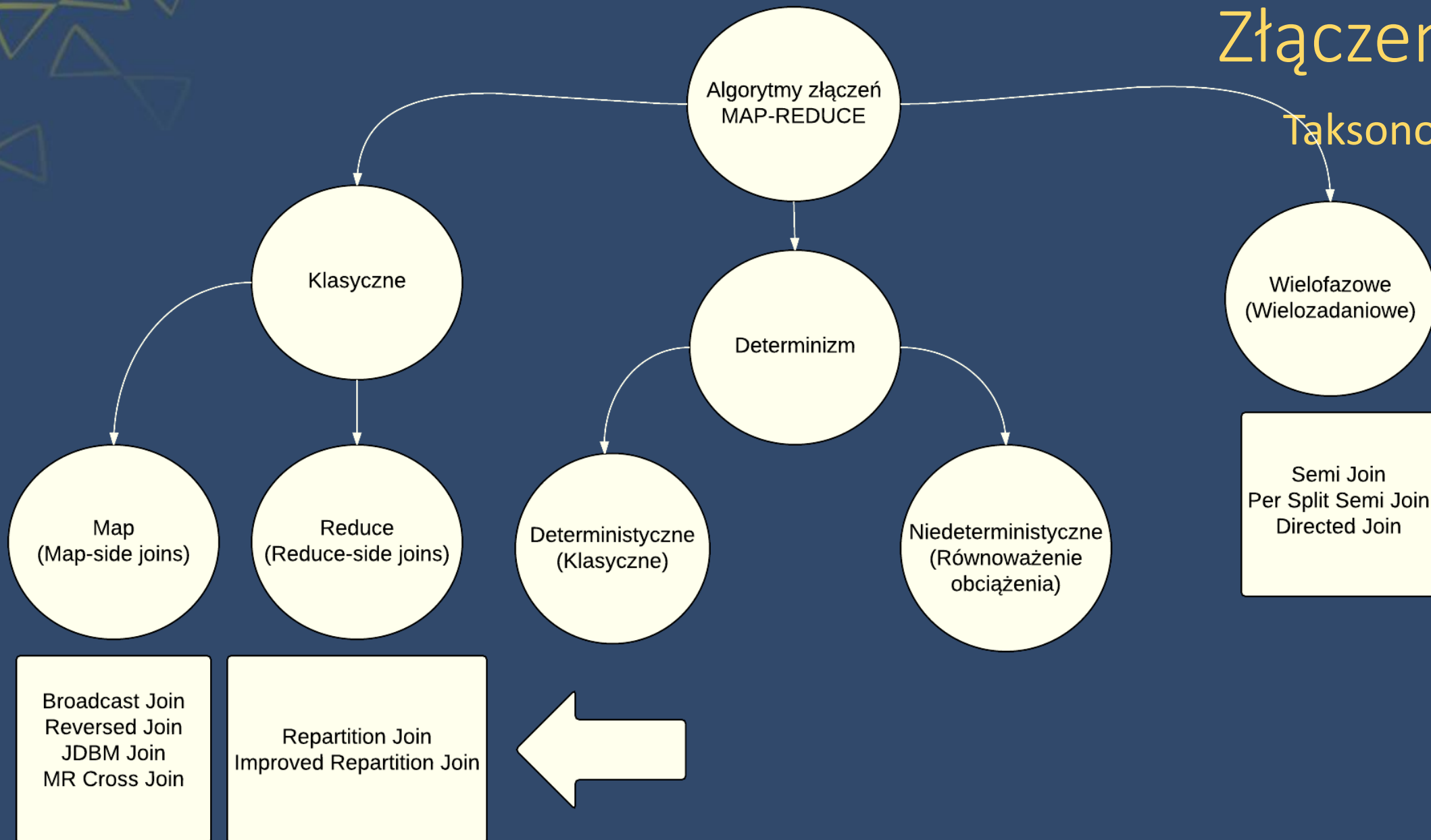
INNER JOIN R_3 ON θ_2

....

INNER JOIN R_N ON θ_{N-1}

Złączenia

Taksonomia



Złączenia

Klasyczne -> Map-Side Joins -> Broadcast Join

Inne nazwy: Map-Side Join, Replicated Join

Koncepcja:

- Jeden (lub kilka) plików rozesłać (rozgłaszać) do wszystkich Mapperów za pomocą mechanizmu *Distributed Cache* (rozgłoszone pliki nie są traktowane jako wejściowe)
- W Mapperze:
 - Wczytać cały plik do pamięci
 - Zbudować HashMap'ę w oparciu o klucz obcy
 - Dla każdego wczytanego rekordu pliku wejściowego znaleźć wpis w HashMapie
 - Wyemitować dowolnie wczytany rekord + rekord z Hashmap'y

Złączenia

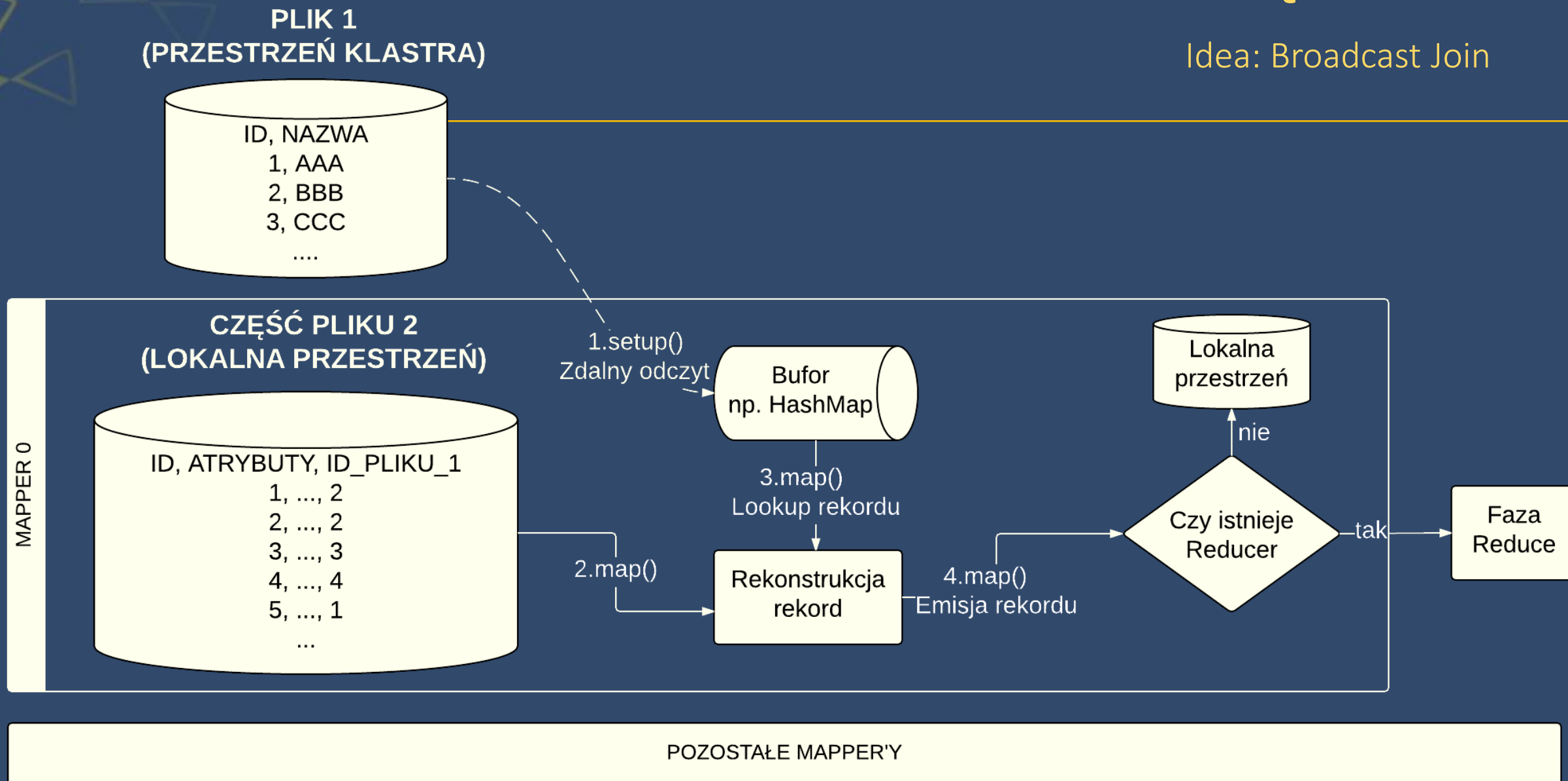
Klasyczne -> Map-Side Joins -> Broadcast Join

Realizacja:

- Przy konfiguracji: `addLocalFiles(Configuration conf, String str)` – dodanie plików do cache
- W metodzie `setup()` Mapper'a:
 - `Path[] path = DistributedCache.getLocalCacheFiles(Configuration conf)` – pobranie ścieżek plików cache
 - `File file = new File(path[?])` – odczyt pliku

Złączenia

Idea: Broadcast Join



Złączenia

Klasyczne -> Map-Side Joins -> Broadcast Join

Zalety:

- Równozłączenie – HashMap'a + Hash Join
- Złączenie theta – dowolna kolekcja + Inner Loop
- Cache'owany plik zostaje na węźle (przez jakiś czas)
- Brak fazy shuffle
- Wielozłączenia

Wady:

- Wszystkie węzły muszą pobrać plik
- OutOfMemoryException dla dużych plików
- Brak fazy Reduce: brak sortowania, liczba plików wejściowych równa liczbie użytych Mapperów

Złączenia

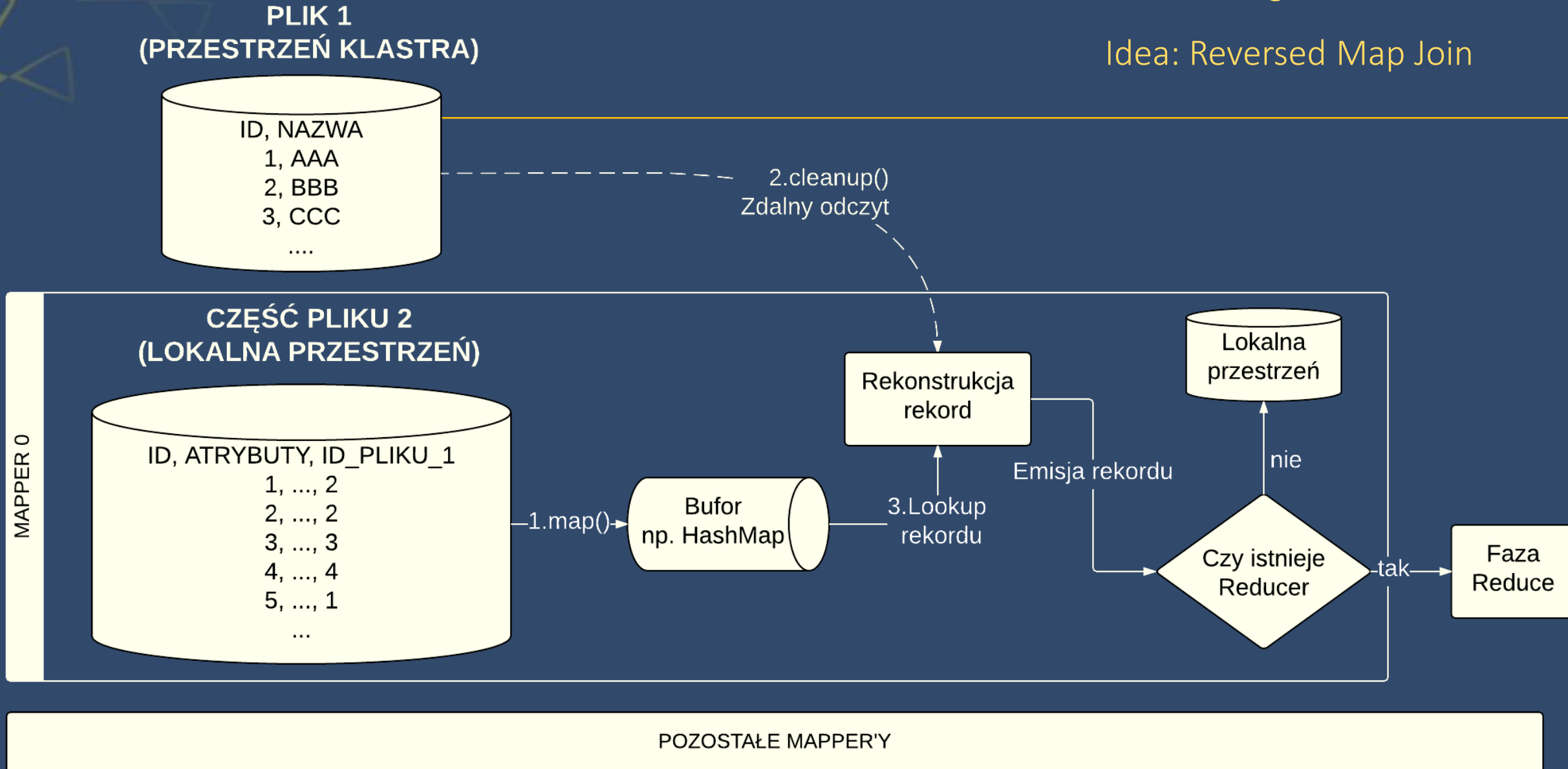
Klasyczne -> Map-Side Joins -> JDBM/Reversed Map Join

Modyfikacje:

- JDBM Join:
 - wykorzystanie kolekcji z biblioteki 'jdbm:jdbm:1.0' – usuwa problem związany z OOM, ale spada wydajność
- Reversed Map Join:
 - setup() – nic nie robi
 - map() – cache'uje dane wejściowe
 - cleanup() – wczytuje rozgłaszany plik i łączy dane
 - W 99% przypadków usuwa OOM, ale łamie model (map() nie emituje rekordu)

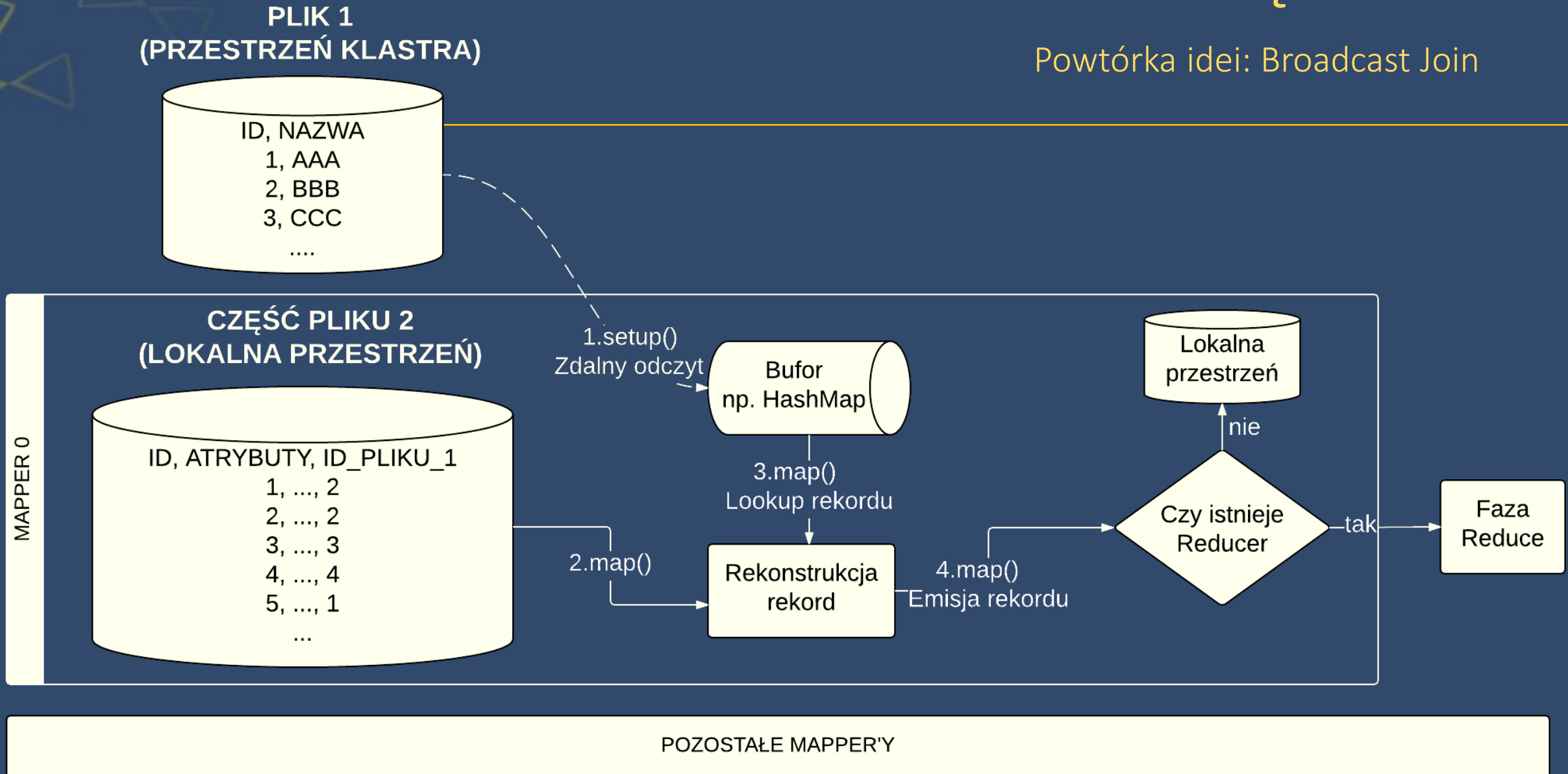
Złączenia

Idea: Reversed Map Join



Złączenia

Powtórka idei: Broadcast Join



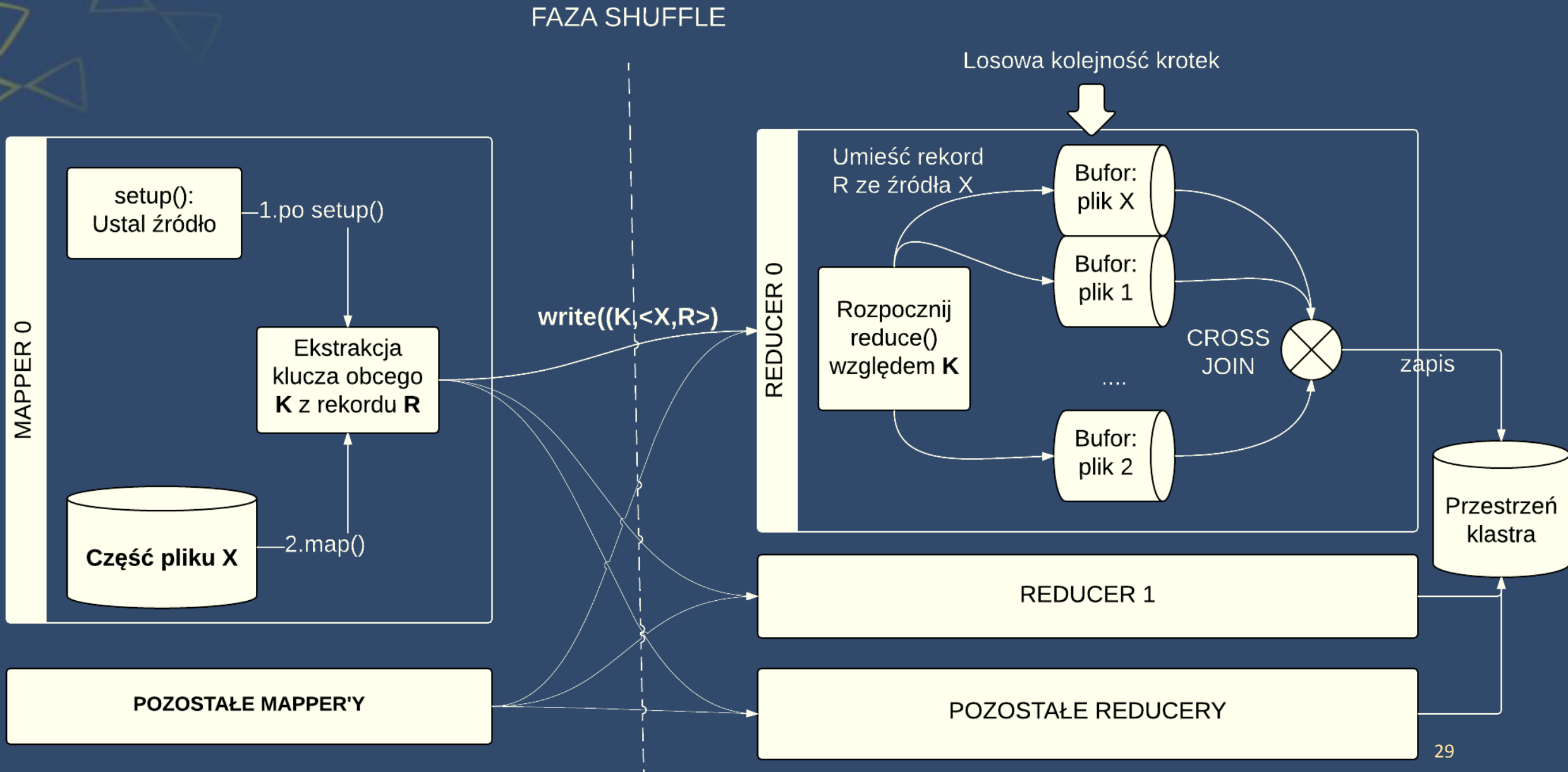
Złączenia

Klasyczne -> Reduce-Side Joins -> Repartition Join

Koncepcja:

- Każdy plik wejściowy (relacje) jest obsługiwany przez osobną implementację Mappera – konfiguracja za pomocą klasy *MultipleInputs*
- Mapper: emituje parę <klucz obcy, <źródło, rekord>>
- Reducer:
 - Buforuje rekordy ze względu na źródło
 - Liczy produkt kartezjański pomiędzy buforami (np. po wywołaniu *hasNext()*)

Idea: Repartition Join



Złączenia

Klasyczne -> Reduce-Side Joins -> Repartition Join

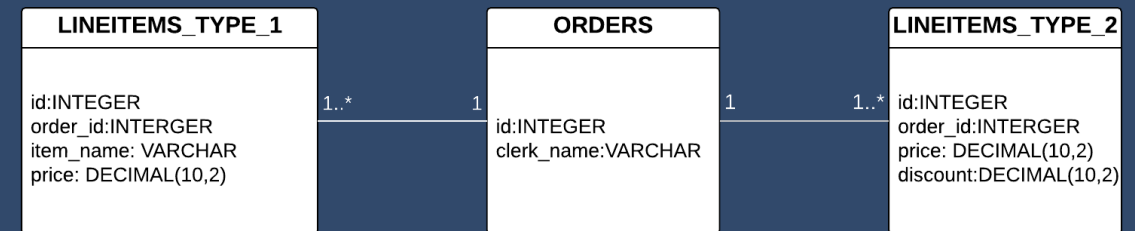
Zalety:

- Wpasowany w model programistyczny
- Wielozłącze dla schematu

odwrotnej gwiazdy

Wady:

- Tylko równozłączenie
- Złączenie bez filtrowania wymaga przestania wszystkich danych
- Buforowanie wszystkich rekordów – miejsce na OOM
- „Krzywe” dystrybucje wartości klucza obcego
- Wielozłącze pomiędzy źródłami danych o różnych kluczach obcych wymaga wielu Job’ów, a to wymaga **Shuffle, Zapisu Reducer’a i Rozstawienia**



Złączenia

Dystrybucje klucza

Istnieje szereg metod przeciwdziałających nierównym dystrybucjom kluczy obcych. IMHO nieprzydatnych ze względu:

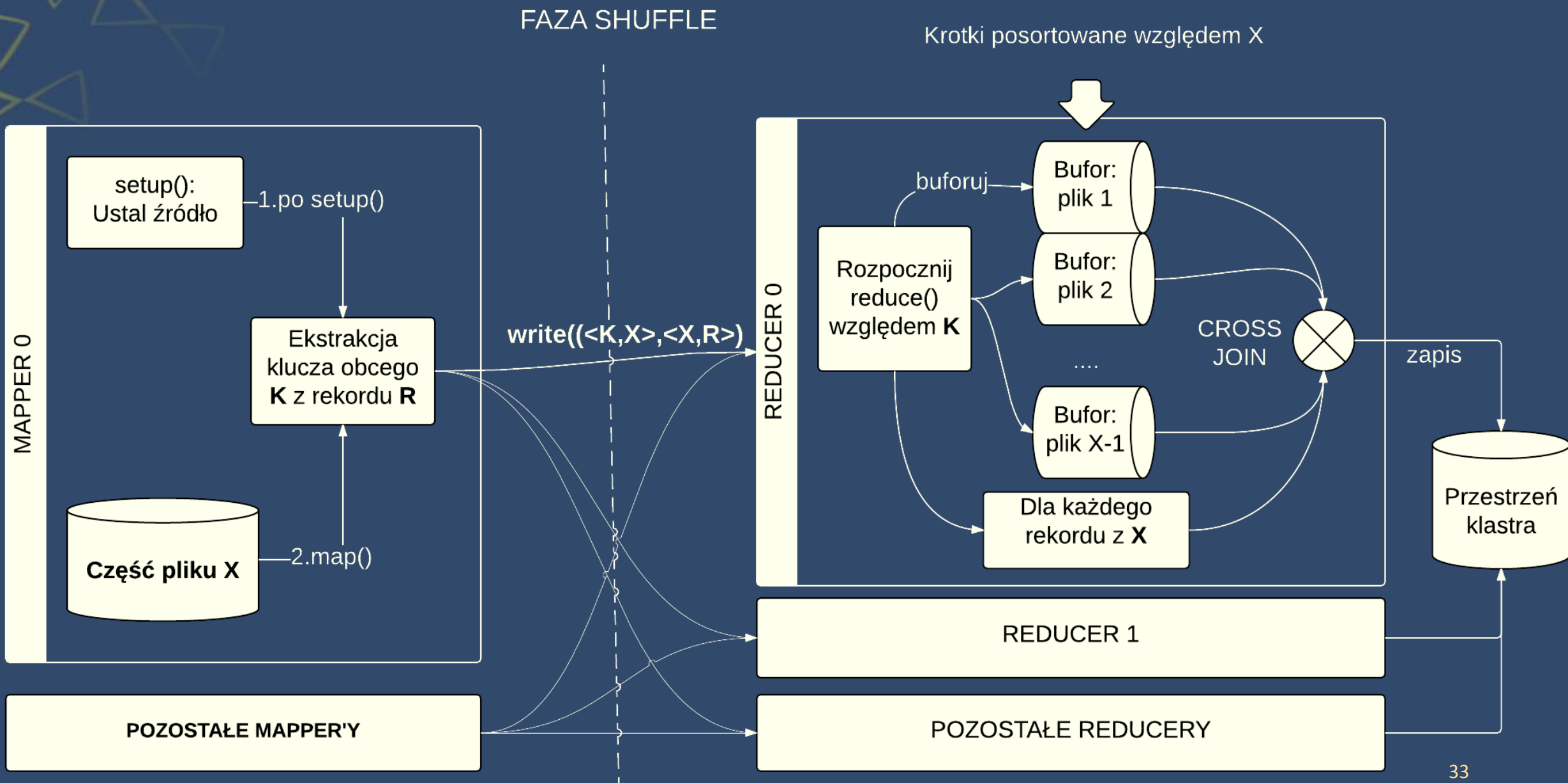
- Map Reduce wykorzystujemy do analizy danych, a nie żeby policzyć złączenie
- Najczęściej znormalizowane są dane słownikowe (np. Miasto) nie mające wpływu na wynik ciekawej dla nas agregacji np. sumy sprzedaży w mieście – możemy redukować (lokalnie) wg. Kluczy obcych
- Ergo: pod każdym kluczem znajdzie się co najwyżej 1 – LICZBA_MAPPERÓW rekordów (przy zaleceniu, że każdy Reduktor powinien przetwarzać min 2 GB)
- Dystrybucja klucza obcego przestaje mieć znaczenie
- Dołożymy do tego filtrowanie po stronie Mappera

Złączenia

Klasyczne -> Reduce-Side Joins -> Improved Repartition Join

- Optymalizacja polegająca na emitowaniu w Mapperach pary:
 << klucz obcy , źródło>>, <źródło, rekord>>
 - Posortowanie po <klucz obcy , źródło>>, tak by najpierw wczytane były rekordy pochodzące z najmniej liczego źródła
 - Grupowanie po <klucz obcy> <- GroupingComparator
 - Dzielenie po <klucz obcy> <- Partitioner
 - Buforowanie rekordów z wszystkich, oprócz ostatniego źródła
 - Dla ostatniego źródła:
 - Wczytujemy rekord
 - Liczymy złączenie w locie
- ! Wydaje mi się że w nowym API może zadziałać na << klucz obcy , źródło>>, rekord>

Idea: Improved Repartition Join



Złączenia

Klasyczne -> Reduce-Side Joins -> Improved Repartition Join

Zalety i wady w zasadzie takie same jak dla Repartition Join, oprócz:

Zalety:

- Buforowanie $N - 1$ łączonych źródeł danych
- Przetwarzanie krotek z ostatniego źródła produkuje wynik w locie

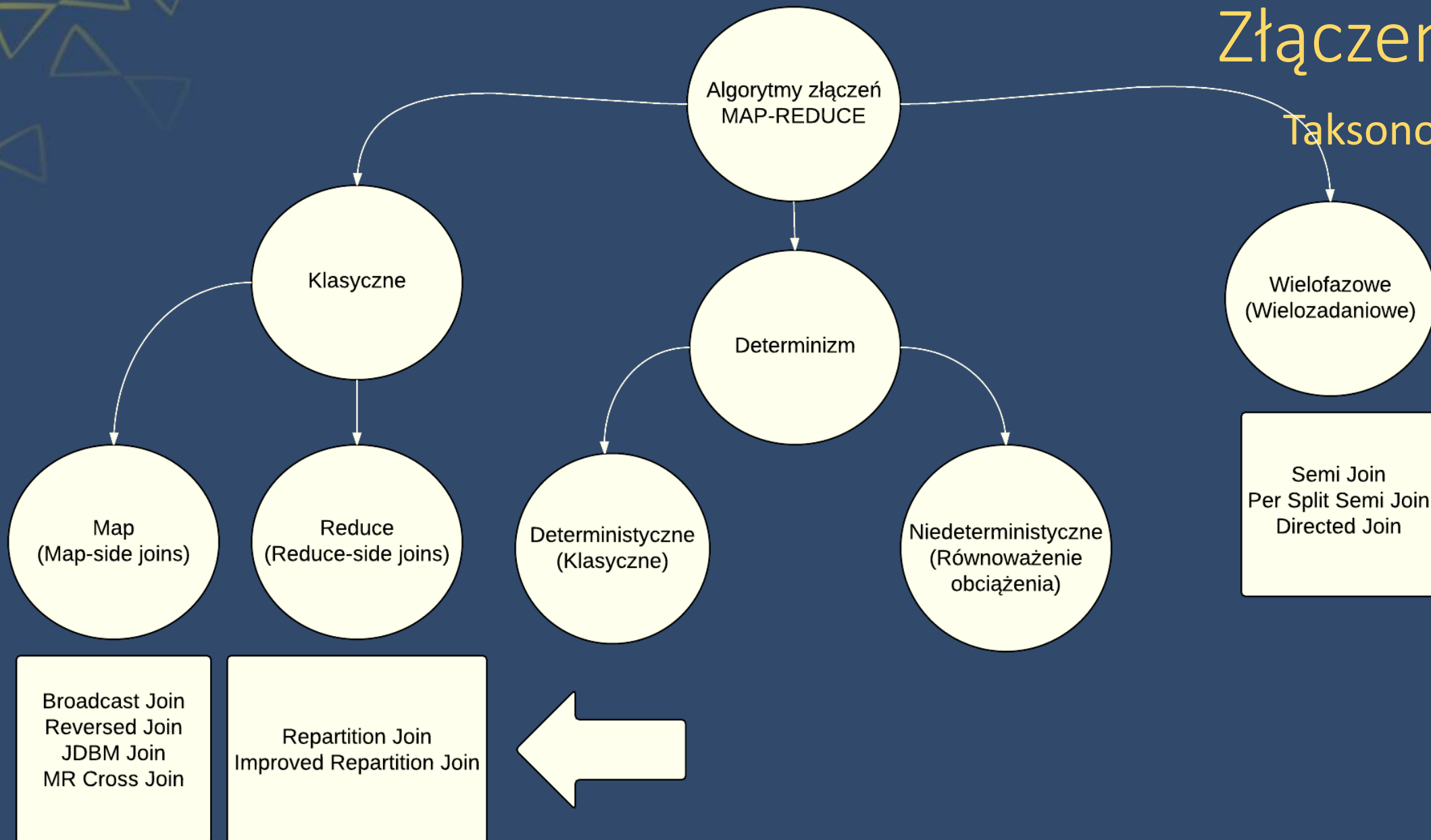
Wady:

- Kolejność przekazywanych parametrów ma znaczenie
- Redundancja wysłania danych dot. źródła
- Spróbujcie napisać do tego kod ogólny

-Ale Hive to robi
-Nie, nie robi

Złączenia

Taksonomia



Złączenia

Wielofazowe -> Directed Join


- Nazywany też: Composite Join*
- Exploit „efektów ubocznych” wywołania programów Map-Reduce
- Eksperyment:
 - Każdy plik ładujemy do Hadoop’a i osobno dla każdego uruchamiamy program
 - Mapper emituje rekordy grupując po kluczu obcym
 - Wykorzystujemy IdentityReducer – przepisuje wszystko co dostał
- Wynik:
 - Każdy plik podzielony na liczbę mniejszych plików równą liczbie wykorzystanych Reducer’ów
 - Każdy plik X-tego Reducera o nazwie part-r-0000X posiada ten sam zakres kluczy obcych
 - Wyniki posortowane po kluczach obcych

* Dobrze opisany w: MapReduce Design Patterns, Donald Miner & Adam Shook

Złączenia

Wielofazowe -> Directed Join

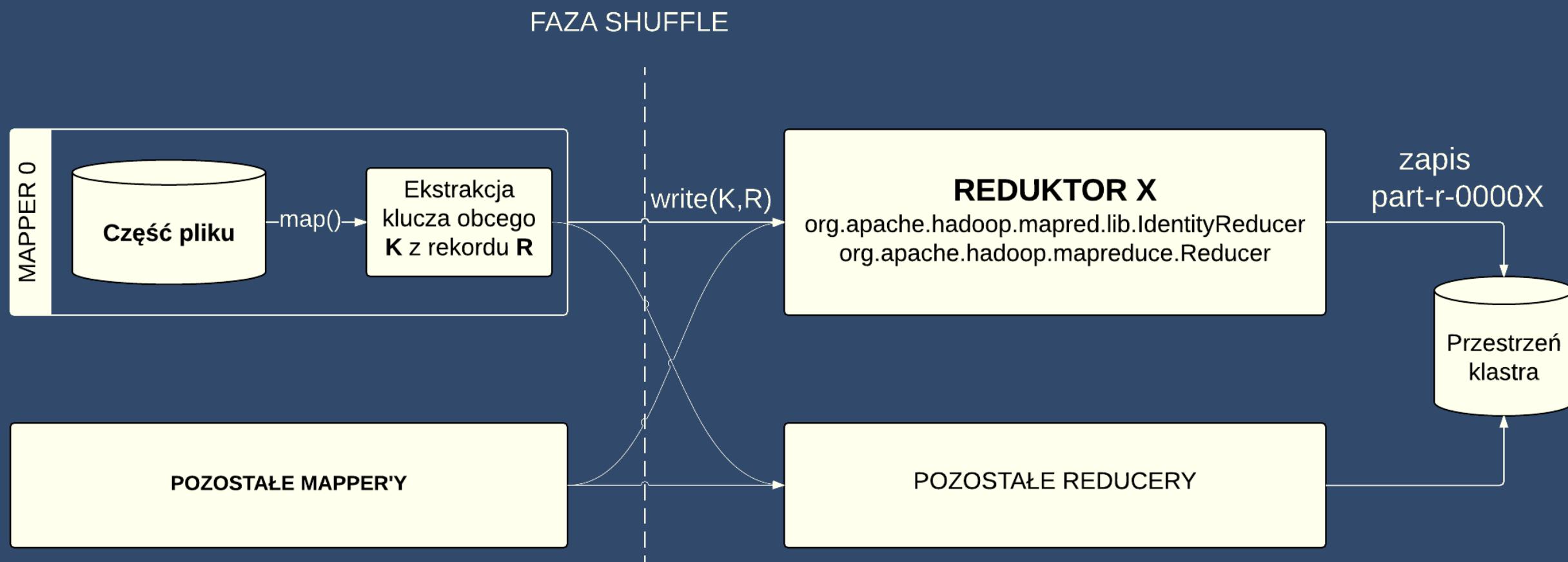
- Zakładając że do przetwarzania plików wykorzystany został ten sam Partitioner – dla każdego k-tego pliku danych ze źródła 1 można wykonać Map Join dla k-tego pliku danych ze źródła 2
- W teorii można to zautomatyzować
- Trzeba dopilnować żeby Mapper przetworzył cały plik
- To rozwiązanie jest już zaimplementowane w pakiecie (Wymaga dodatkowej konfiguracji m.in.: CompositeInputFormat):
org.apache.hadoop.contrib.utils.join



Biblioteka jest napisana w starym API – nie testowałem jej – dostałem po twarzy ClassCastException, zaimplementowałem sam i zamiotłem sprawę pod dywan

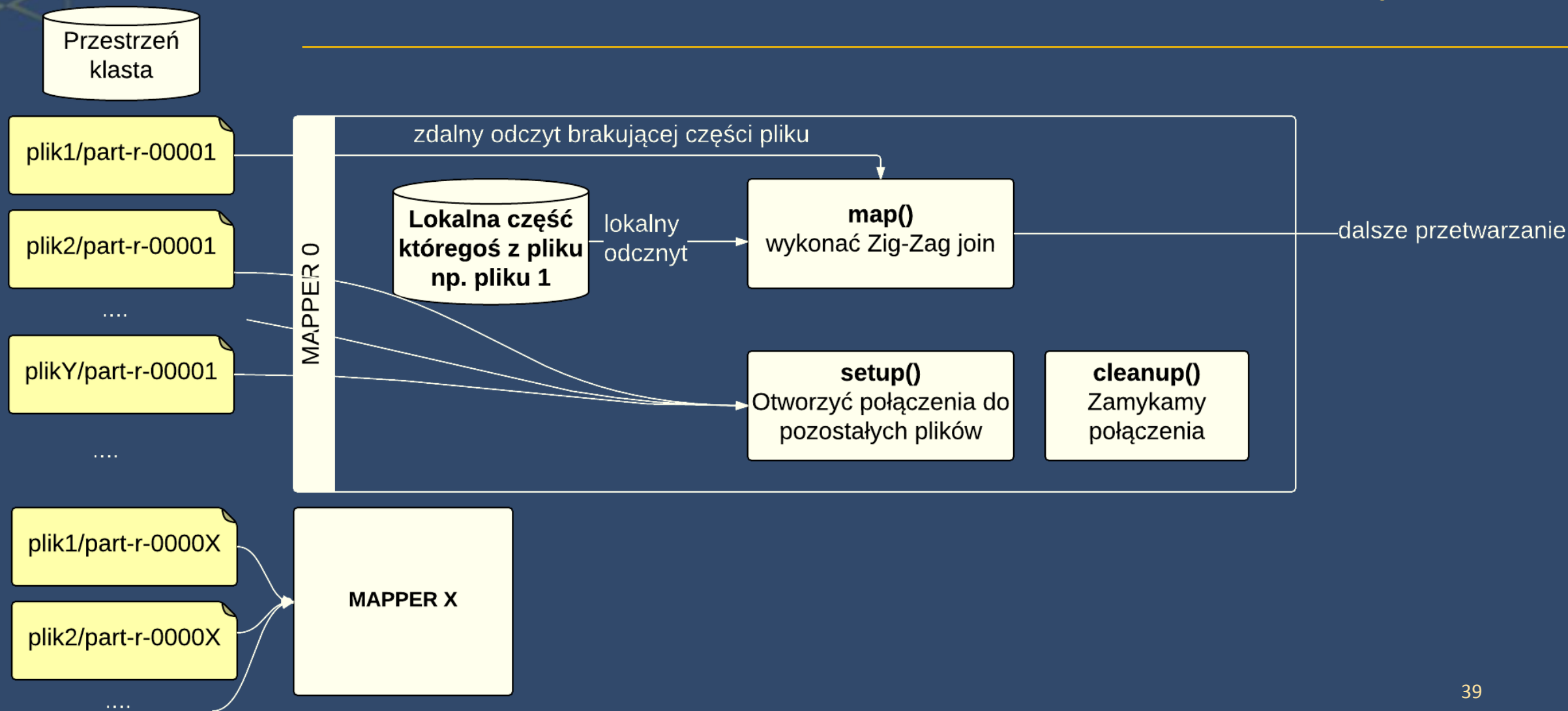
Złączenia

Idea: Directed Join - preprocessing



Złączenia

Idea: Directed Join - złączenie



Złączenia

Wielofazowe -> Directed Join

„Atomowe” przetwarzanie pliku można łatwo zrealizować za pomocą:

- Dziedziczenia klasy `FileInputFormat`
- Przeciążenia metody: `isSplittable(){ return true;}`

Directed Join można zaimplementować nie na poziomie plików, a tzw. Input Splitów (danych wejściowych Mapper’a)

Przykład znajduje się w książce: *MapReduce Design Patterns*, Donald Miner & Adam Shook

Złączenia

Wielofazowe -> Semi Join

- Dodatkowe Job'y do filtrowania krotek wiszących (na bazie pół-złączeń)
- Dowolność implementacji, np.:
 - Job 1: Wczytujemy 1 plik
 - Map - emitujemy jedynie klucze obce
 - Reduce – usuwamy duplikaty
 - Job 2: Wczytujemy 2 pliki
 - Map – emitujemy klucze obce które pojawiają się w pliku wynikowym z Job'a 1
 - Reduce – jak w Repartition Join
- Istnieje zaawansowana wersja zwana Per-Split Semi-Join, gdzie filtrowanie odbywa się nie na poziomie pliku (wszystkich Mapperów), a pojedynczego Mapper'a



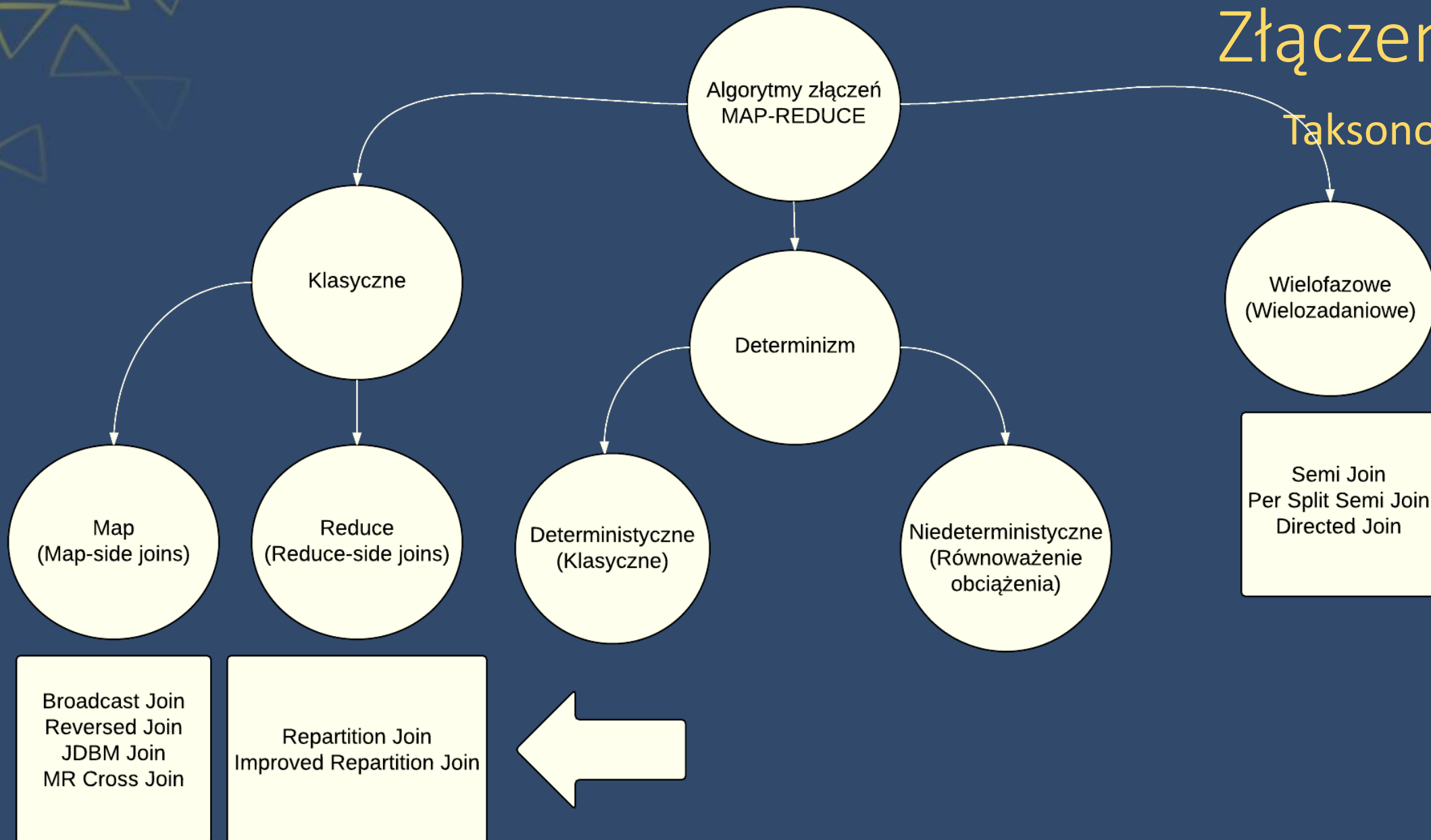
Złączenia

Wielofazowe -> Semi Join

- Celem jest usunięcie krotek wiszących
- W warunkach „produkcyjnych” Combiner i tak zredukuje ich liczbę – więc po co się trudzić
- Nie znam framework’a który używa jakąkolwiek z technik wielofazowych tu wymienionych

Złączenia

Taksonomia



Złączenia

Load Balancing Approach: idea

Założmy że mamy dwa pliki CSV – „plik_1” i „plik_2”... wypiszmy pierwszy z nich rekord po rekordzie w kolumnie.

Plik 1

1,aaa,2.2

2,bbb,4.3

3,ccc,5.1

4,ddd,1.1

5,aaa,4.2

6,ccc,2.1

Złączenia

Load Balancing Approach: idea

Teraz wypiszmy drugi z nich identycznie.

Plik 1

1,aaa,2.2

2,bbb,4.3

3,ccc,5.1

4,ddd,1.1

5,aaa,4.2

6,ccc,2.1

Plik 2

1,aaa,bbbb,...

1,bbb,aaaa,...

2,ccc,bbbb,...

3,ddd,bbbb,...

3,fff,aaa,...

4,ggg,bbbb,...

4,yyy,bbbb,...

4,aaa,bbbb,...

Złączenia

Load Balancing Approach: idea

Obróćmy – zauważmy że powstaje prostokątna przestrzeń

Plik 1

1,aaa,2.2
2,bbb,4.3
3,ccc,5.1
4,ddd,1.1
5,aaa,4.2
6,ccc,2.1

Plik 2

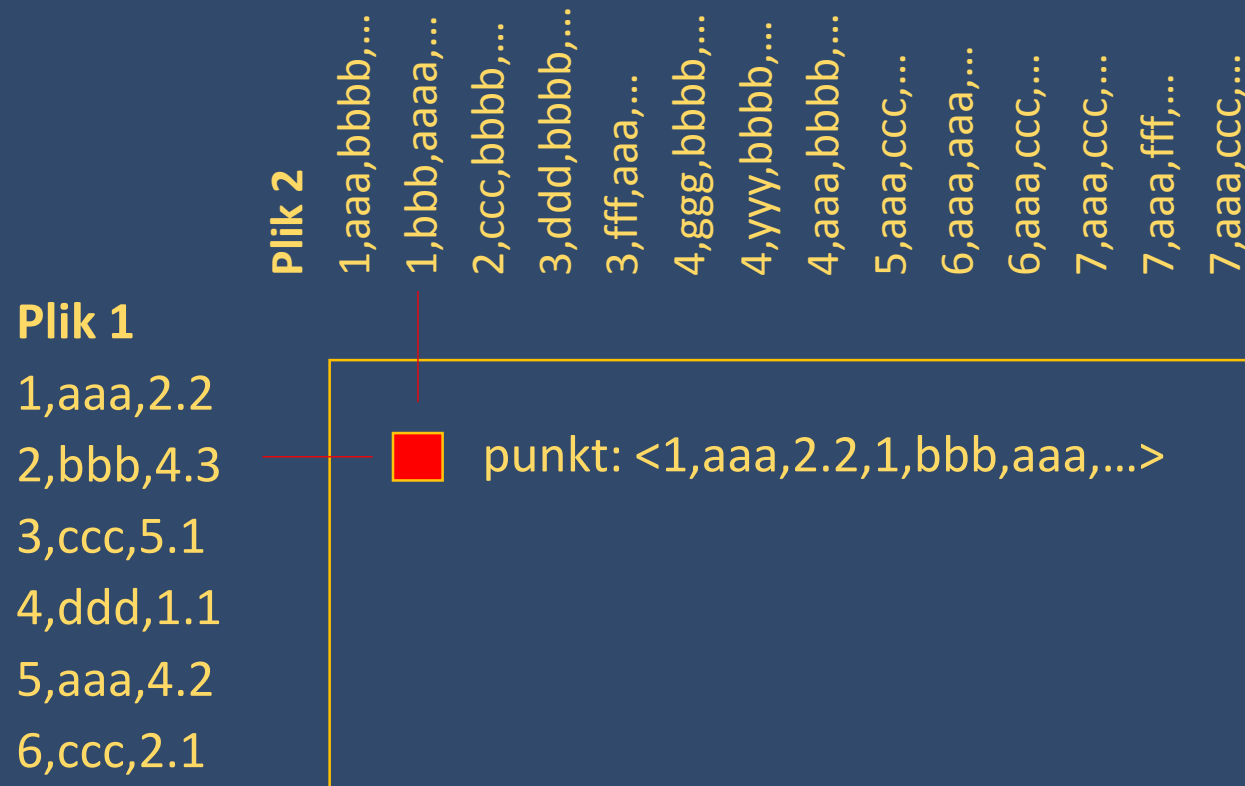
1,aaa,bbbb,...
1,bbb,aaaa,...
2,ccc,bbbb,...
3,ddd,bbbb,...
3,fff,aaa,...
4,ggg,bbbb,...
4,yyy,bbbb,...
4,aaa,bbbb,...
5,aaa,ccc,...
6,aaa,aaa,...
6,aaa,ccc,...
7,aaa,ccc,...
7,aaa,fff,...
7,aaa,ccc,...



Złączenia

Load Balancing Approach: idea

Obróćmy – zauważmy że powstaje prostokątna przestrzeń



Złączenia

Load Balancing Approach: idea

Wyberzmy pewną liczbę np. 6. Podzielmy przestrzeń na taką liczbę prostokątów i ponumerujmy powstałe prostokąty (od 0)

Plik 1

1,aaa,2.2
2,bbb,4.3
3,ccc,5.1
4,ddd,1.1
5,aaa,4.2
6,ccc,2.1

Plik 2

1,aaa,bbb,bbb,...
1,bbb,aaaa,...
2,ccc,bbb,bbb,...
3,ddd,bbb,bbb,...
3,fff,aaa,...
4,ggg,bbb,bbb,...
4,yyy,bbb,bbb,...
4,aaa,bbb,bbb,...
5,aaa,ccc,...
6,aaa,aaa,...
6,aaa,ccc,...
7,aaa,ccc,...
7,aaa,fff,...
7,aaa,ccc,...

0	1	2
3	4	5

Złączenia

Load Balancing Approach: idea

Niech prostokąty oznaczają dane które „wpadają” do Reduktora o podanym identyfikatorze. Zbiór wszystkich Reducerów umożliwia policzenie Cross Join’a. Przy takim podziale każdy Reduktor wykonuje podobną liczbę iteracji/przetwarza podobną liczbę krotek. Pytanie jak ustalić miejsce podziału.

Plik 1

1,aaa,2.2
2,bbb,4.3
3,ccc,5.1
4,ddd,1.1
5,aaa,4.2
6,ccc,2.1

Plik 2

1,aaa,bbbb,...
1,bbb,aaaa,...
2,ccc,bbbb,...
3,ddd,bbbb,...
3,fff,aaa,...
4,ggg,bbbb,...
4,yyy,bbbb,...
4,aaa,bbbb,...
5,aaa,ccc,...
6,aaa,aaa,...
6,aaa,ccc,...
7,aaa,ccc,...
7,aaa,fff,...
7,aaa,ccc,...

0	1	2
3	4	5

Złączenia

Load Balancing Approach: idea

W tym celu usuńmy z naszego diagramu abstrakcję plików.

0	1	2
3	4	5

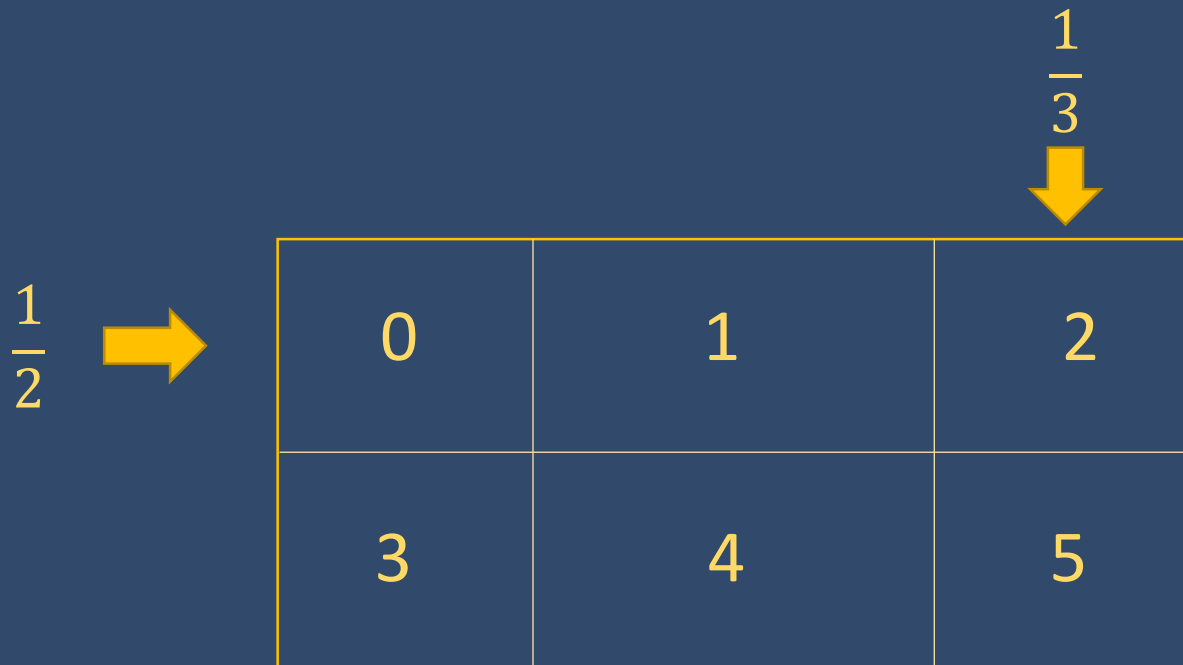
Złączenia

Load Balancing Approach: idea

Przyporządkujemy równomiernie prawdopodobieństwo umieszczenia rekordu w Reduktorach kolumny/wiersza :

- kolumny mają prawdopodobieństwo $\frac{1}{3}$
- wiersze mają prawdopodobieństwo $\frac{1}{2}$

Każda komórka (Reduktor) tym samym otrzyma około $\frac{1}{6}$ wszystkich danych



Złączenia

Load Balancing Approach: idea

Teraz implementujemy Mappery dla pliku 1 tak by:

- Losowały liczbę ze zbioru {0,1}
- Jeśli wylosowano 0, emitujemy rekordy <0,wiersz>, <1,wiersz> i <2,wiersz>
- Jeśli wylosowano 1, emitujemy rekordy <3,wiersz>, <4,wiersz> i <5,wiersz>

Z kolei Mapper dla pliku 2 analogicznie na kolumnach:

- Losujemy liczbę ze zbioru {0,1,2}
- Jeśli wylosowano 0, emitujemy rekordy <0,wiersz>, <3,wiersz>
- Itd....

Złączenia

Load Balancing Approach: spostrzeżenia

- Co potem?
 - Literatura mówi: W `reduce()` „użyj ulubionego algorytmu do Cross Join’a”
 - Każdy rekord z Cross Joina filtrujemy
- Podejście daje się uogólnić na przestrzeń o dowolnej liczbie wymiarów
- W przykładzie podział był 2 wiersze i 3 kolumny - [2,3] – co oznacza że rekordy z pliku 1 będą 2-krotnie zreplikowane, a z pliku 2 – 3-krotnie
- Shuffle jako faza najbardziej czasochłonna – a my wysyłamy więcej danych
- Filtrowanie Cross Join’a - żałosna selektywność selekcji
- Podział wpływa bezpośrednio na efektywność rozwiązania – da się mierzyć

Model programistyczny

Load Balancing Approach: Mój stosunek do modelu Map Reduce

1. „W rozważaniach teoretycznych przyjmuje się następujący model”
2. Programiści łamiący model
 - Podejście burzy determinizm
 - Programista przepisuje Partitioner i Mapper - ogólnie rzecz biorąc - odseparowano programistę od systemu, a ten przepisuje jego funkcjonalność
 - Grupowanie, a w zasadzie dzielenie danych ze względu na sztuczny klucz – liczba wywołań reduce() jest równa liczbie Reduce'ów
 - Niestety to jedyny sposób żeby wykonać pełnoprawne Theta-złącze w modelu Map-Reduce (ale nie w jego rozszerzeniach)

Złączenia

Load Balancing Approach: optymalizacja

Dla zadanej liczby Reduktorów R trzeba znaleźć optymalny podział plików

Zakładając dwa pliki o rozmiarach F_1, F_2 szukamy takich liczb całkowitych p_1 i p_2 które minimalizują koszt fazy Shuffle:

$$c(p_1, p_2) = R \left(\frac{F_1}{p_1} + \frac{F_2}{p_2} \right), \text{ gdzie } R = p_1 p_2$$

Po uogólnieniu na N plików:

$$c(p_1, \dots, p_N) = R \sum_i^N \frac{F_i}{p_i}, \text{ przy ograniczeniu } R = \prod_i^N p_i$$

Graficznie: kwadraty, sześciany, hipersześciany itd.



Złączenia

Antywzorce

Czas na Plot Twist
Antywzorce



Złączenia

Antywzorce

Applications not using a higher-level interface such as Pig unless really necessary

Źródło: <https://developer.yahoo.com/blogs/hadoop/apache-hadoop-best-practices-anti-patterns-465.html>

Złączenia

Frameworki

Frameworki na licencji Apache:

- Hive
- Pig
- Drill
- Tajo
- Spark
- Inne...



Złączenia

Frameworki

Języki:

- **Hive - HiveQL**
- Pig – PigLatin <- proceduralny
- Drill - SQL
- Tajo – TSQL (TajoSQL)
- Spark – Spark SQL

Nie są mi znane języki operujące w ramach Hadoopa które potrafią wykonać złączenie nierównościowe

Złączenia

HiveQL - Tabele zewnętrzne – jak SQL

```
CREATE EXTERNAL TABLE `default.TABLEA`(  
    `A1` int ,  
    `B1` int ,  
    `C1` string ,  
    `D1` double,  
    `E1` string,  
    `F1` string,  
    `G1` string,  
    `H1` string,  
    `I1` string)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY "|" "  
STORED AS TEXTFILE  
LOCATION '/user/peczwy/data/1n/VOLUME/orders';
```

Złączenia

HiveQL

```
SELECT page_views.*  
FROM page_views  
WHERE page_views.date >= '2008-03-01'  
        AND page_views.date <= '2008-03-31',
```

Złączenia

HiveQL – rodzaje

Złączenia:

1. `x JOIN y ON c`
2. `x {LEFT|RIGHT|FULL} [OUTER] JOIN y ON c`
3. `x LEFT SEMI JOIN y ON c`
4. `x CROSS JOIN y`

Złączenia

HiveQL – Złączenia nierównościowe

```
SELECT a.val, b.val, c.val  
FROM a CROSS JOIN b  
WHERE ABS(a.key - b.key) < 5;
```

Kontrowersyjna wydajność, ale działa

Złączenia

HiveQL - Annotacje

STREAMTABLE – strumieniowanie podanej tabeli (domyślnie najbardziej z *prawej* strony:

```
SELECT /*+ STREAMTABLE(b) */ a.val, b.val, c.val  
FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)
```

MAPJOIN – zaproszenie do złączenia po stronie Mapperów

```
SELECT /*+ MAPJOIN(b) */ a.key, a.value  
FROM a JOIN b ON a.key = b.key
```


"Hive does not support join conditions that are not equality conditions as it is **very difficult** to express such conditions as a map/reduce job.,”*

* <https://cwiki.apache.org/Hive/languagemanual-joins.html>

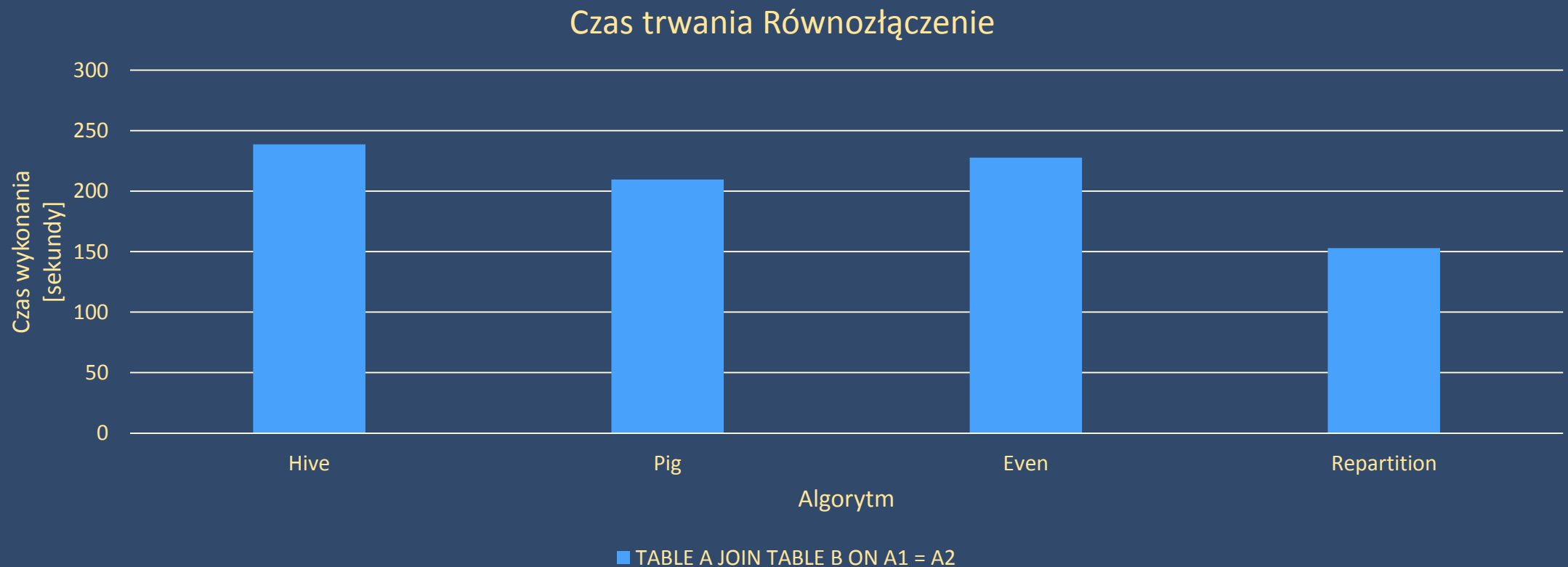
* <http://grokbase.com/t/hive/user/123dnszatz/non-equality-joins>

Wydajność

- Wydajność rozumiana jako czas utylizacji klastra – czas wykonania Job'a
- Test:
 - Klaster 5 węzłów (chmura Amazona)
 - Złączenie dwóch tabel, relacja 1-N
 - Zbiór danych TPC-H – 1 GB
 - Obfuskacja danych (kolumn)
 - Dwa zapytania
 - Równozłączenie: `TABLEA JOIN TABLEB ON A1 = A2`
 - Theta-złączenie: `E1 <= date_add(K2, 30) AND E1 >= date_sub(K2, 30)`

Wydajność

Równozłączenie

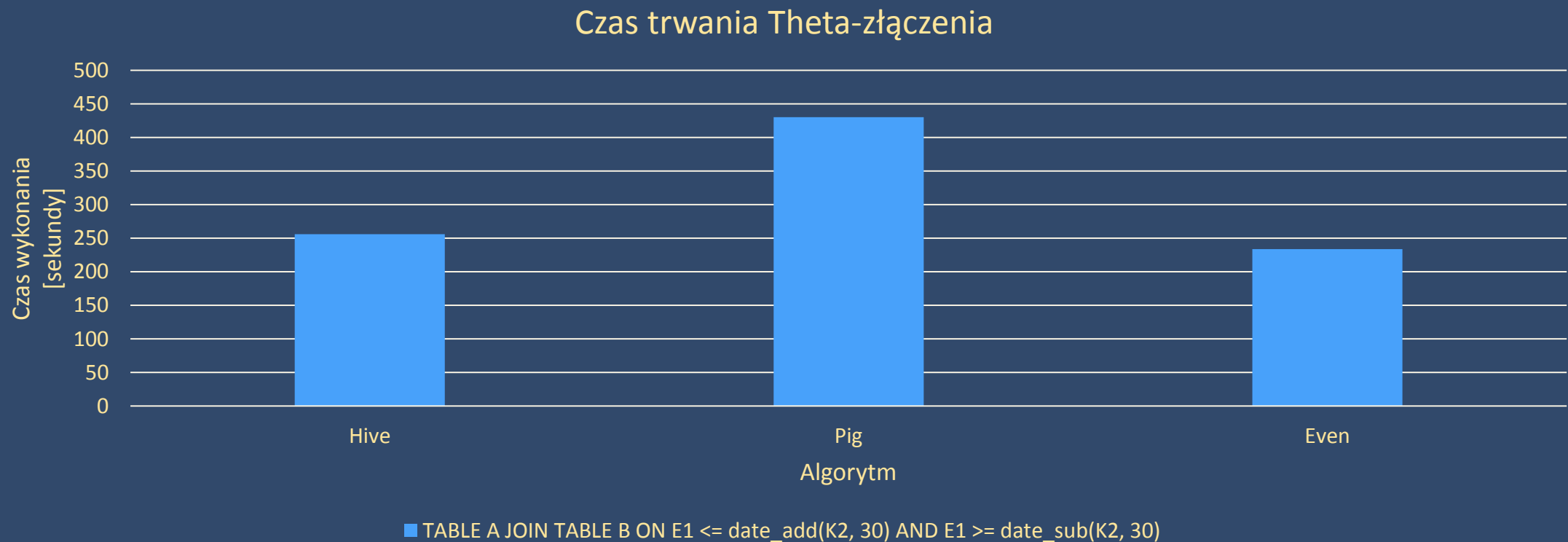


* <https://cwiki.apache.org/Hive/languagemanual-joins.html>

* <http://grokbase.com/t/hive/user/123dnszatz/non-equality-joins>

Wydajność

Theta-złączenie



* <https://cwiki.apache.org/Hive/languagemanual-joins.html>

* <http://grokbase.com/t/hive/user/123dnszatz/non-equality-joins>

Podsumowanie

- Nad każdym klastrem powinien wisieć Framework (co najmniej jeden)
- Kodzimy w ostateczności
- Złączenia nierównościowe w ogólnej postaci **nie** są możliwe do wyrażenia za pomocą grupowania – agregacji
- Systemy Map Reduce NIE są kompletne obliczeniowo i NIE powinny być używane jak *złoty młotek*
- Istnieją inne modele programistyczne np. PACT, który rozszerza liczbę możliwych faz

Materiały

- Hadoop The Definitive Guide 4th edition, Tom White
- Hive reference: [link](#)
- MapReduce Design Patterns, Donald Miner & Adam Shook
- MapR Academy: [link](#)
- Hortonworks VM: [link](#)
- Join algorithms in Map/Reduce, Maciej Penar, praca magisterska, Politechnika Wrocławska



Dziękuję za uwagę

Pytania