

# Obiektowe bazy danych

## Implementacja obiektowych baz danych

# Plan wykładu

- ▶ Architektura obiektowych systemów baz danych uwzględniająca specyficzny model przetwarzania
- ▶ Fizyczne zarządzanie obiektami
- ▶ Wydajny dostęp do kolekcji obiektów
- ▶ Obsługa atrybutów i związków wielowartościowych

# **Specyfika obiektowego modelu danych**

- ▶ Identyfikatory obiektów
- ▶ Powiązania referencyjne między obiektami
- ▶ Atrybuty wielowartościowe
- ▶ Wyrażenia ścieżkowe
- ▶ Hierarchie rozszerzeń klas
- ▶ Duże obiekty

# Nowy wzorzec przetwarzania danych

Klasyczne zastosowania baz danych:

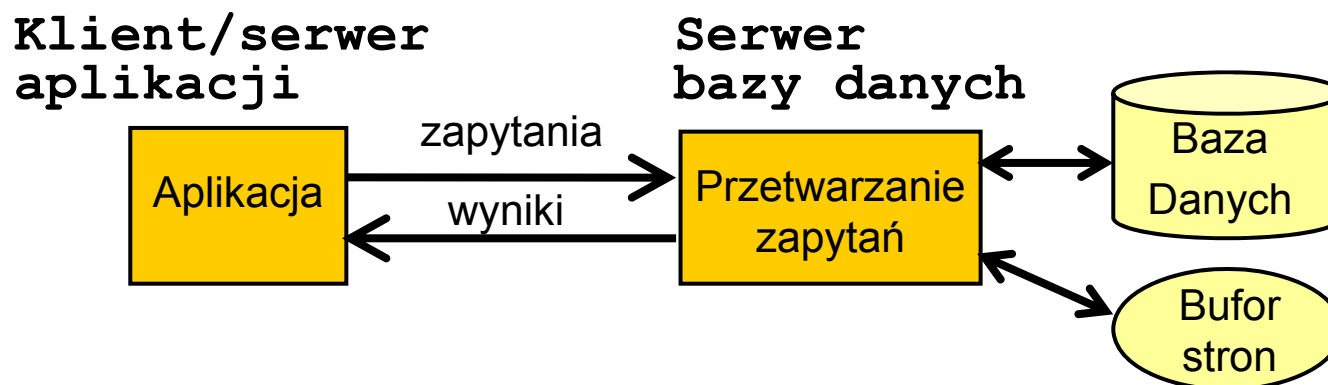
- ▶ W ramach pojedynczej transakcji - proste, jednoetapowe, jednokrotne przetwarzanie niewielkich, homogenicznych kolekcji danych

Zastosowania baz danych typowe dla OODBs:

- ▶ W ramach pojedynczej sesji - złożone, wieloetapowe, wielokrotne przetwarzanie dużych kolekcji złożonych strukturalnie danych

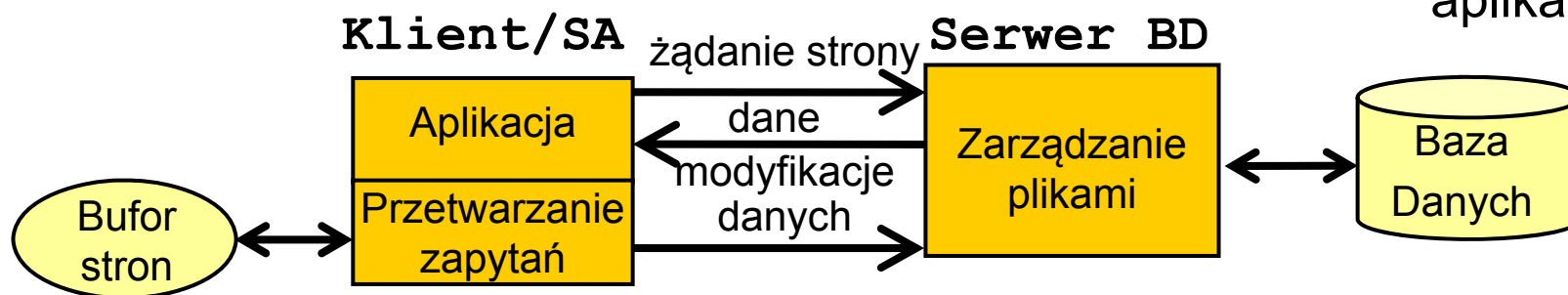
# Architektura obiektowych baz danych

Klasyczna architektura klient-serwer – przesyłanie wyników zapytań



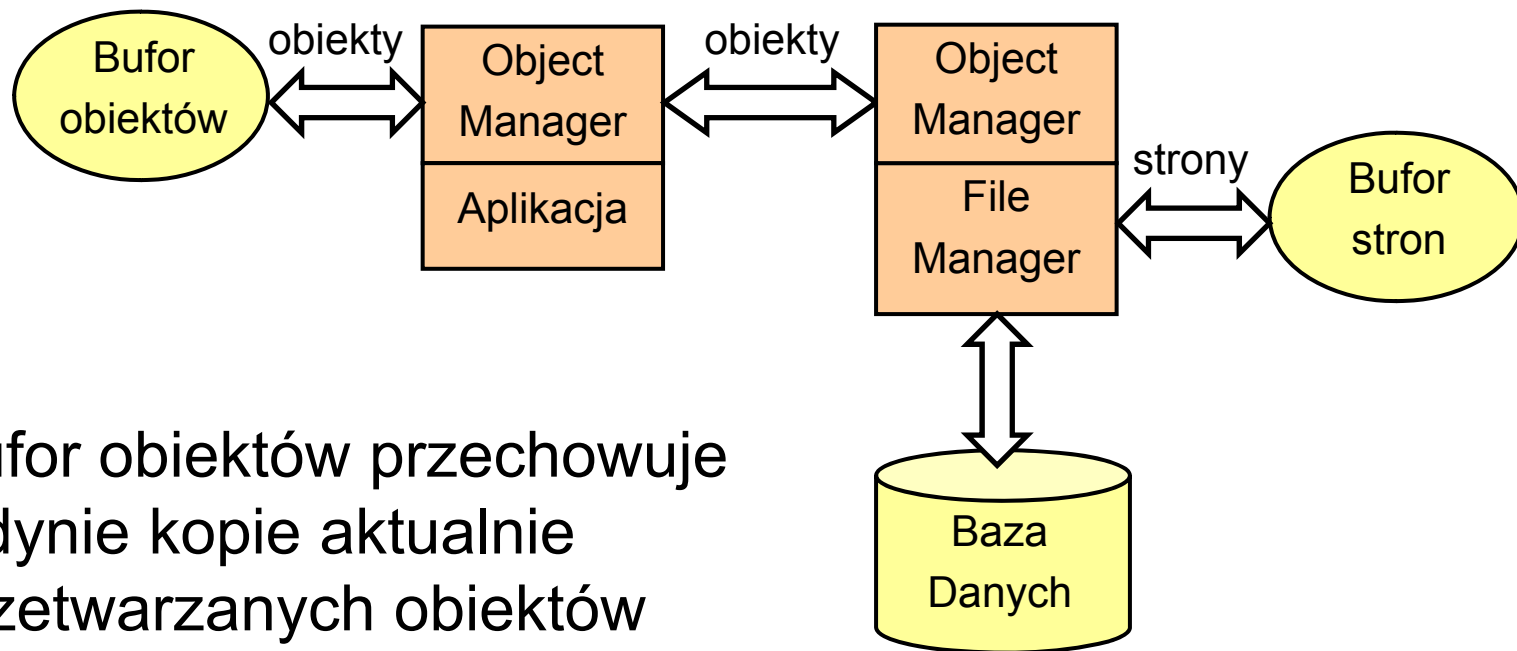
Obiektowa architektura klient-serwer – przesyłanie danych

- Unikanie wielokrotnego przesyłania tych samych danych
- Wykonywanie kodu logiki biznesowej na stacjach klientów/serwerach aplikacji



# Buforowanie obiektów

Dla lepszego wykorzystania obszaru buforów, zamiast stron dyskowych buforowane są pojedyncze obiekty

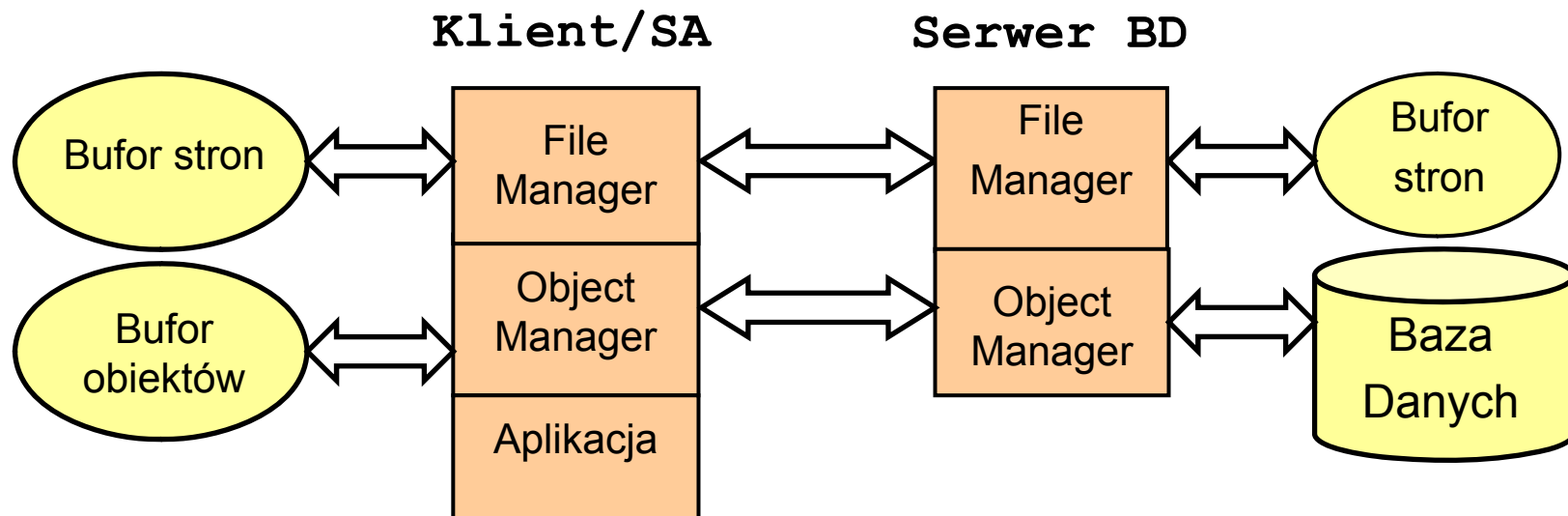


Bufor obiektów przechowuje jedynie kopie aktualnie przetwarzanych obiektów

# Dwupoziomowy bufor bazy danych

Uniezależnia wydajność od rozkładów danych.

- ▶ Bufor stron dla dobrze pogrupowanych danych
- ▶ Bufor obiektów dla słabo pogrupowanych danych.



# **Strategie utrzymania bufora obiektów**

## **Kopiowanie obiektów między buforem stron a buforem obiektów**

- Kopiowanie obiektu z bufora stron do bufora obiektów
  - Natychmiastowe – w momencie pierwszego dostępu (BO)
  - Opóźnione – przed usunięciem strony z bufora (BO+BS)
- Relokacja obiektu do bufora stron
  - Natychmiastowa – po powtórnym załadowaniu strony
  - Opóźniona – po zakończeniu używania obiektu

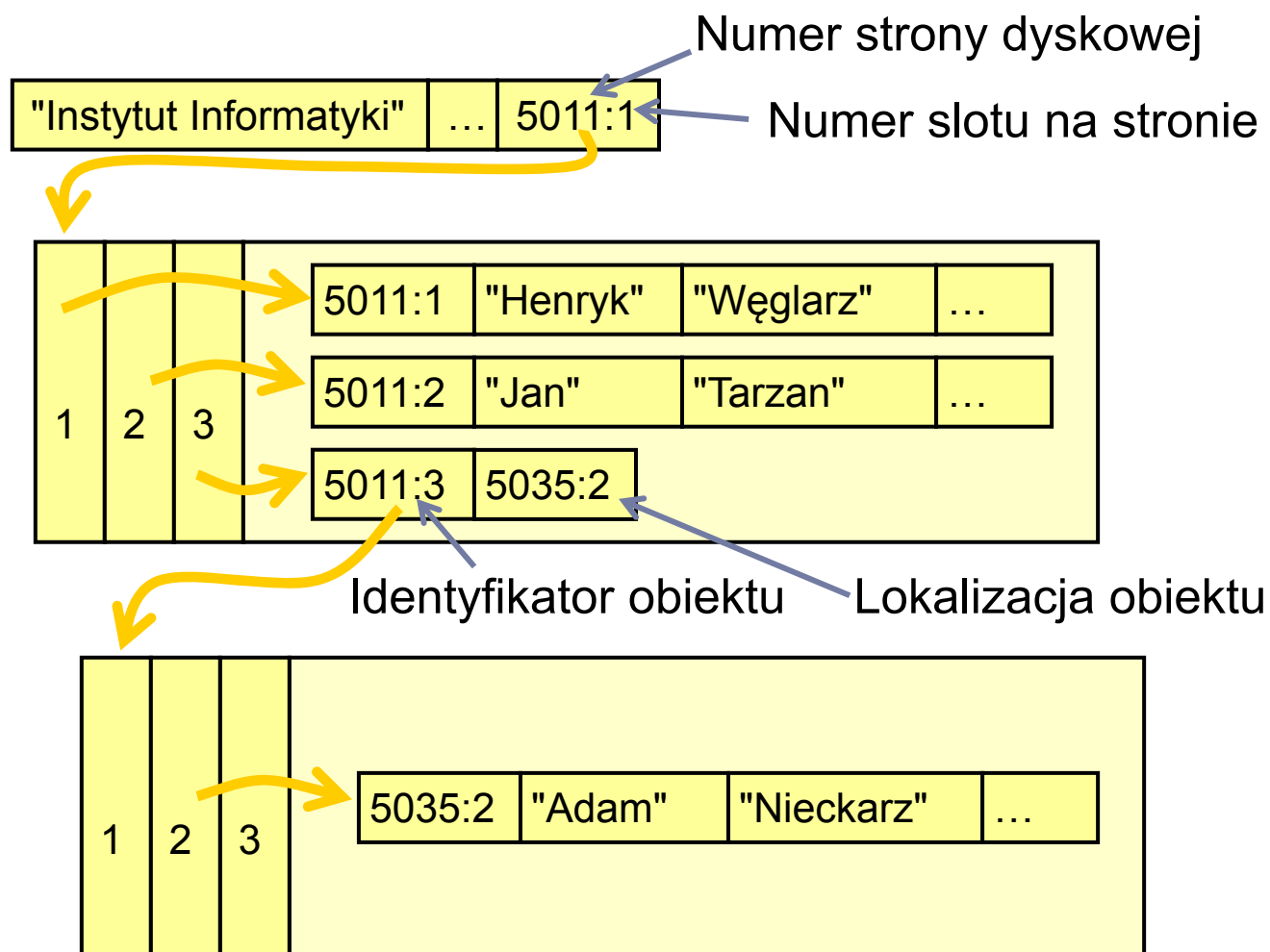
**Gdzie są przetwarzane obiekty (w buforze stron czy obiektów)?**

**Jak duże są bufory stron i bufory obiektów ?**



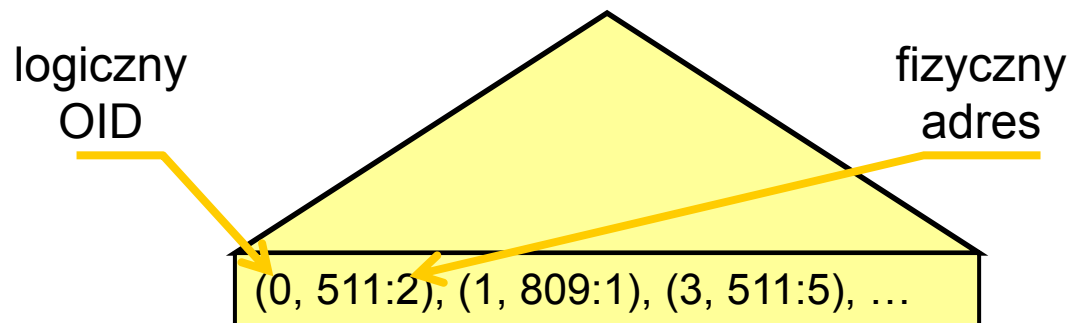
# Implementacja OID

## Fizyczne identyfikatory obiektów



# Logiczne identyfikatory obiektów

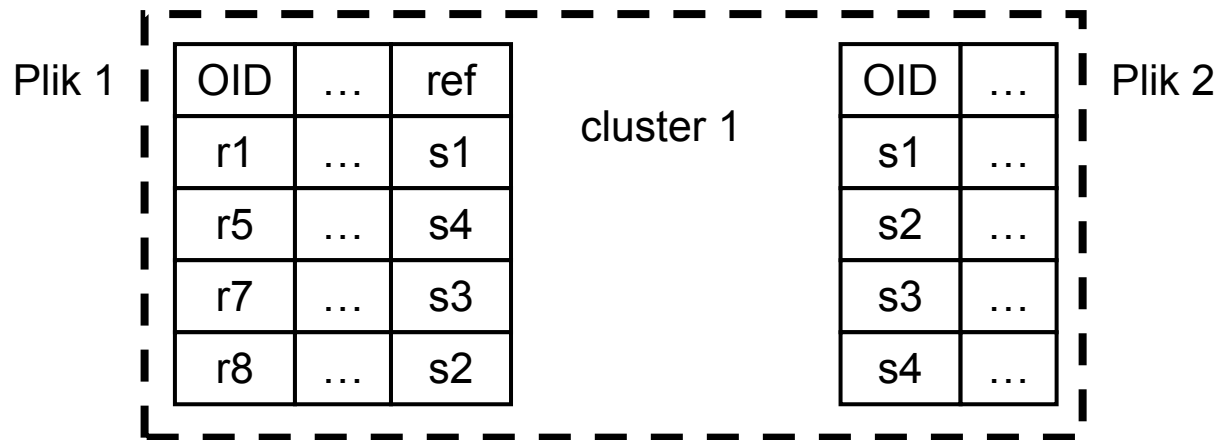
## ► B+-drzewo założone na OID



- tablica haszowa
- odwzorowanie bezpośrednie – logiczny identyfikator jest indeksem w tablicy odwzorowania

# Implementacja operacji połączenia strukturalnego

Nawigacja wzdłuż powiązań referencyjnych jest wydajna dla niewielkich podzbiorów obiektów. Złożoność obliczeniowa tej metody jest liniowa. Jednak dla dużych podzbiorów obiektów liczba operacji dyskowych może być większa niż w metodzie *nested loop*. Wydajna nawigacja wzdłuż referencji wymaga uporządkowania plików danych lub plików zgrupowanych.

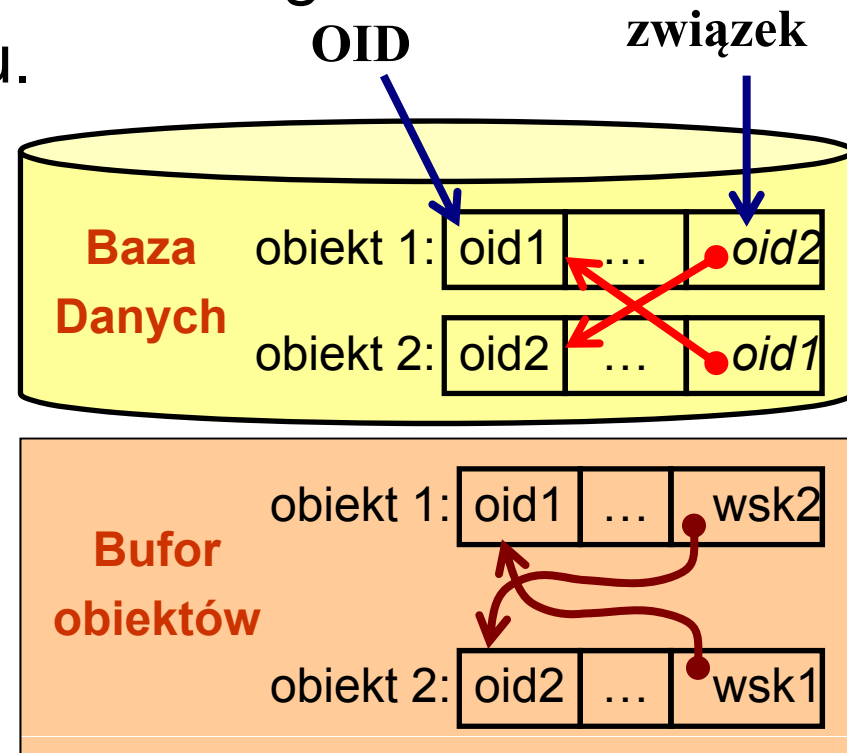


# Transformacja identyfikatorów obiektów

Transformacja identyfikatorów obiektów to zamiana identyfikatora obiektu na adres obiektu w pamięci operacyjnej - opłacalna dla wielokrotnego przetwarzania danego obiektu.

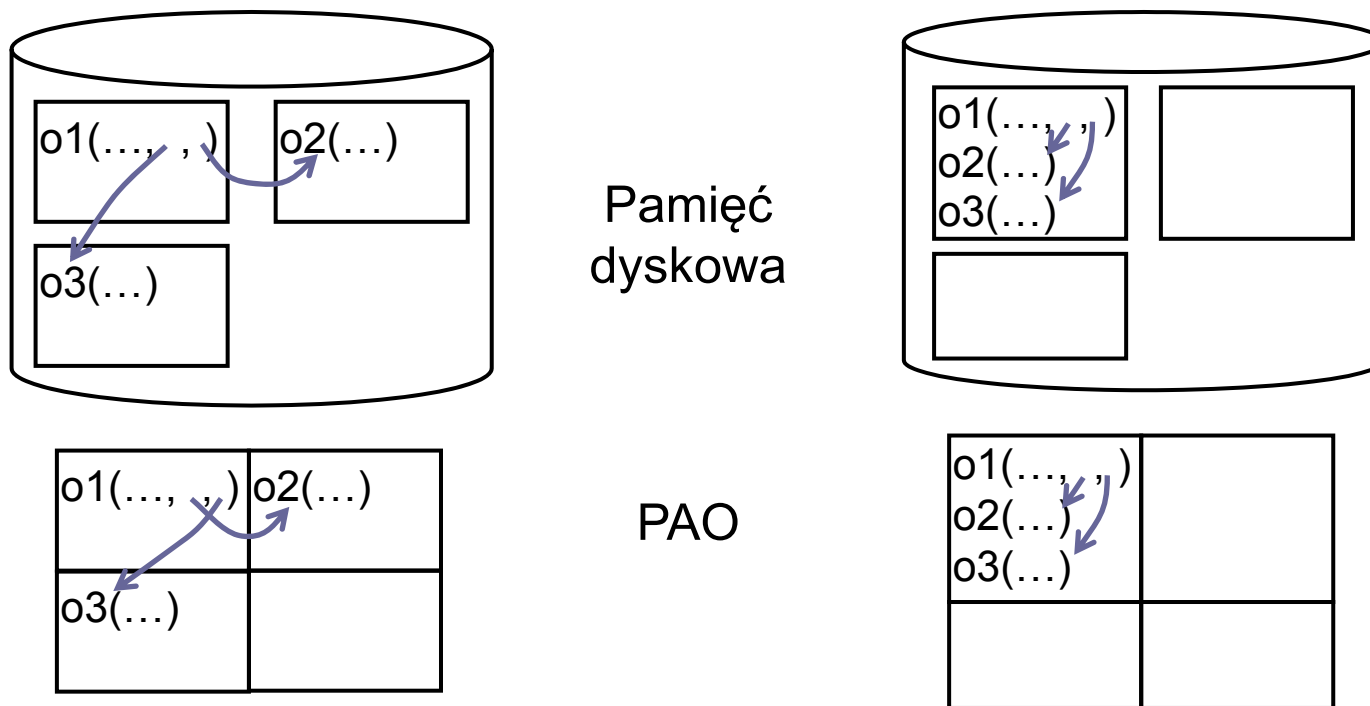
Alternatywy:

- Miejsce transformacji:  
bufor stron ↔ bufor obiektów
- Czas transformacji:  
natychmiastowe ↔ opóźnione
- Sposób transformacji:  
bezpośrednia ↔ pośrednia



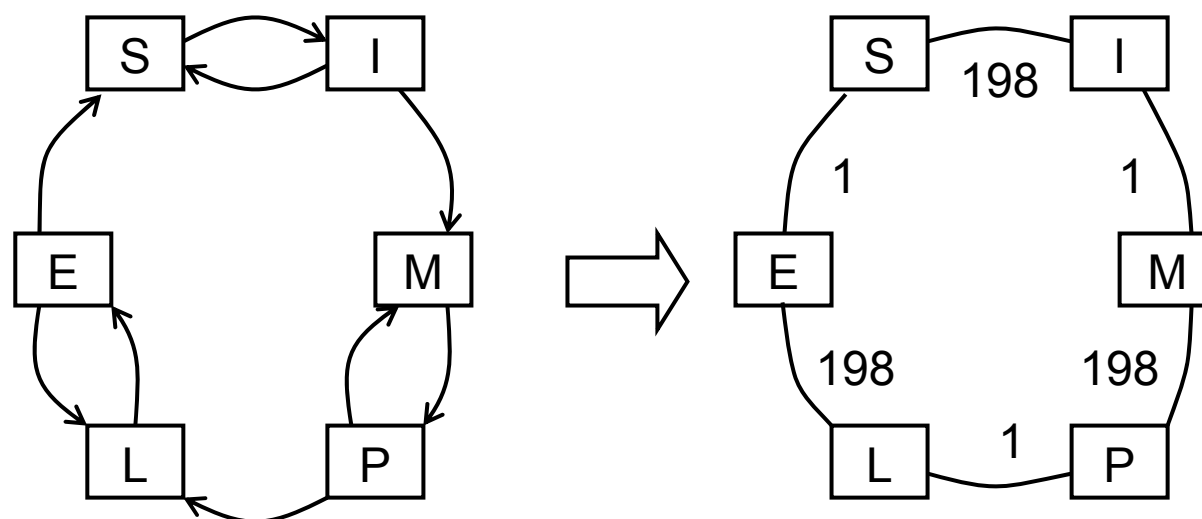
# Grupowanie obiektów na dysku

Celem właściwego grupowania obiektów na stronach dyskowych jest minimalizacja liczby operacji dyskowych i maksymalizacja wykorzystania buforów.



# Grupowanie obiektów na dysku

Problem optymalnej alokacji obiektów w plikach zgrupowanych może być zamodelowany jako problem podziału grafu. Węzły grafu grupowania odpowiadają obiektom bazy danych, krawędzie opisują powiązania w dostępie do obiektów, a wagi krawędzi liczbę kolejnych odwołań.



# Grupowanie obiektów na dysku

Optymalne grupowanie obiektów na strony bazy danych można zamodelować jako podział grafu na podgrafy, zawierające nie więcej niż  $bfr$  węzłów ( $bfr$  jest maksymalną liczbą obiektów, które zmieszczą się na pojedynczej stronie dyskowej), który minimalizuje sumaryczną wagę dzielonych krawędzi. Niestety problem ten należy do klasy silnie NP-trudnych.

# Algorytm podziału grafu grupowania

Heurystyczne rozwiązanie problemu - zachłanny algorytm GGP

**Input:** Graf grupowania GG; **Output:** Lista grup LG

$P_i$  grupa obiektów obejmująca obiekt  $o_i$ ;

$w$  waga krawędzi;

**begin**

Przydziel każdy obiekt do osobnej grupy;

Wstaw utworzone grupy do LG;

Utwórz listę krawędzi LK o postaci:  $(o_i, o_j, w)$ ;

Posortuj listę krawędzi malejąco według wagi;

**foreach**  $(o_i, o_j, w)$  **in** LK **do begin**

**if**  $P_i \neq P_j$  **and**  $P_i \cup P_j$  jest mniejsza od rozmiaru strony  
        przenieś wszystkie obiekty  $P_j$  do  $P_i$ ;

        usuń  $P_j$  z LG;

**end if**;

**end foreach**;

**end**;



# Algorytm podziału grafu grupowania

**Maksymalne wypełnienie stron:**  $b_{fr} = 3$

**Początkowy zbiór pod-grafów:**

$LG = \{P_1 = \{S\}; P_2 = \{I\}; P_3 = \{M\}; P_4 = \{P\}; P_5 = \{L\}; P_6 = \{E\}; \}$

**Lista krawędzi:**

$LK = \{(S,I,198), (I,M,1), (M,P,198), (P,L,1), (L,E,198), (E,S,1)\}$

**Posortowana lista krawędzi:**

$LK_s = \{(S,I,198), (M,P,198), (L,E,198), (I,M,1), (P,L,1), (E,S,1)\}$

1.  $LG = \{P_1 = \{S,I\}; P_3 = \{M\}; P_4 = \{P\}; P_5 = \{L\}; P_6 = \{E\}; \}$

2.  $LG = \{P_1 = \{S,I\}; P_3 = \{M,P\}; P_5 = \{L\}; P_6 = \{E\}; \}$

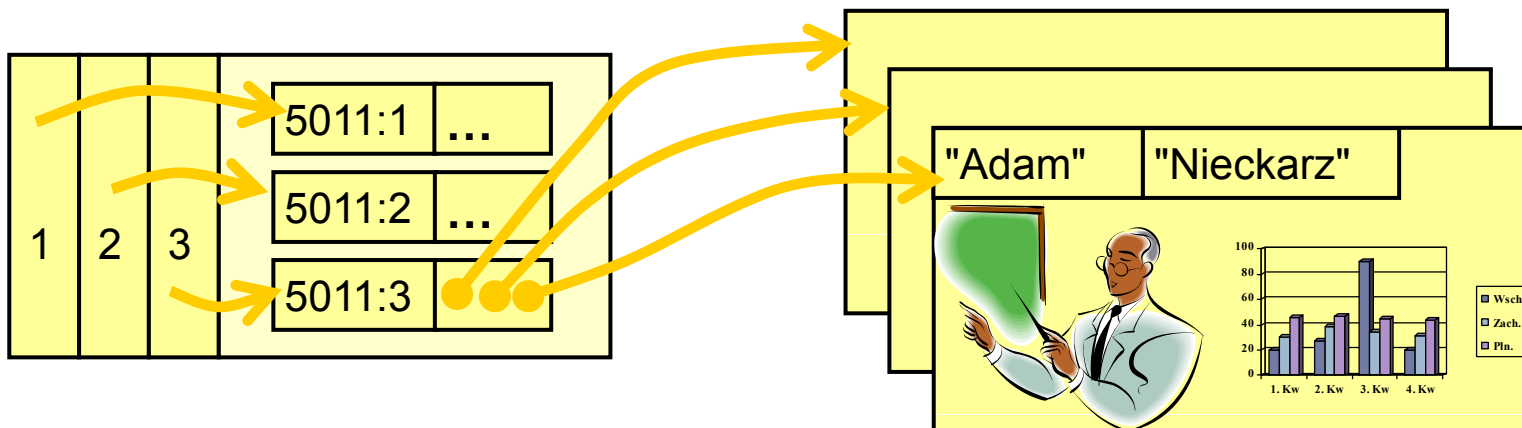
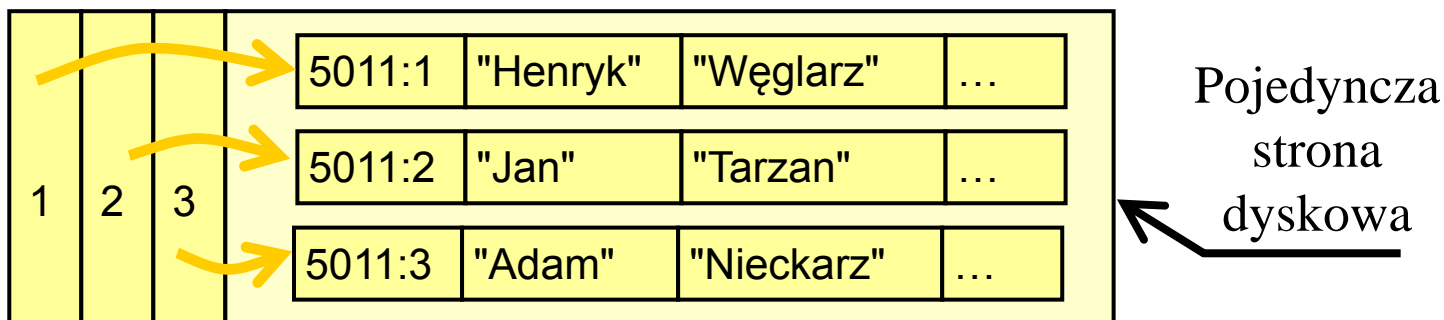
3.  $LG = \{P_1 = \{S,I\}; P_3 = \{M,P\}; P_5 = \{L,E\}\}$

**Ostateczny zbiór pod-grafów:**

$LG = \{P_1 = \{S,I\}; P_3 = \{M,P\}; P_5 = \{L,E\}\}$

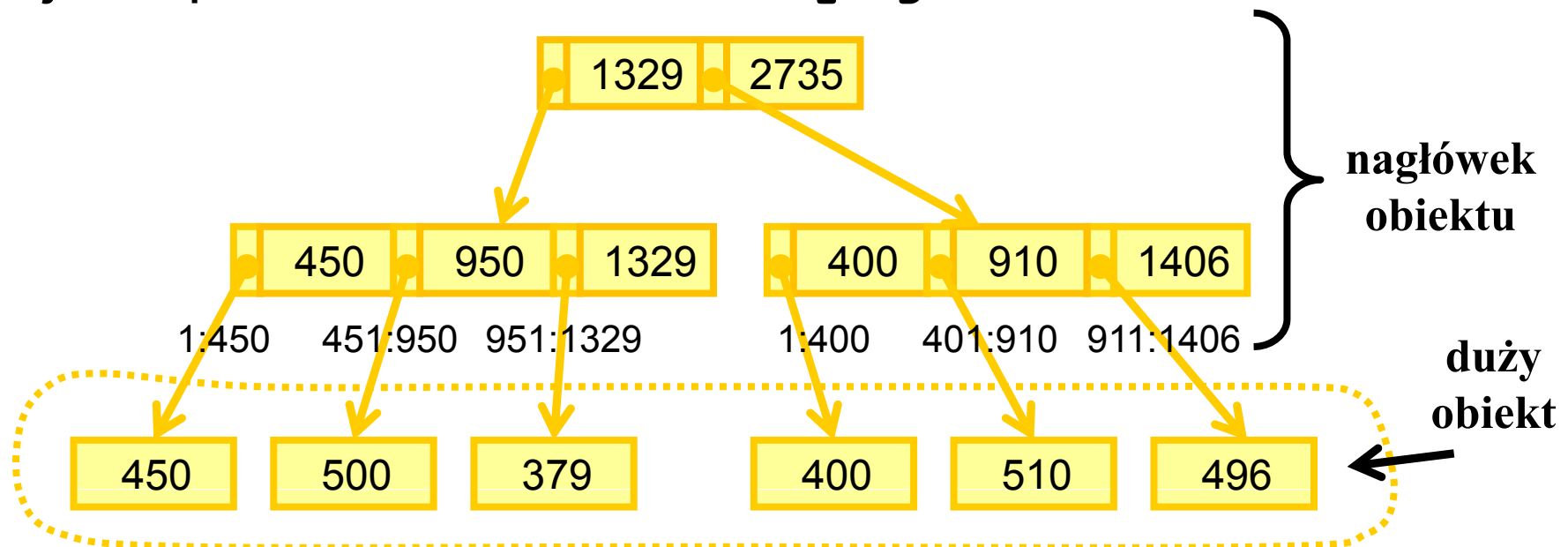
# Składowanie obiektów

Małe obiekty mieszczą się w całości na pojedynczej stronie dyskowej



# Zarządzanie dużymi obiektami

Struktura dużych obiektów musi umożliwiać wydajny dostęp do mniejszych fragmentów obiektów. W tym celu, stosuje się rzadkie B\*-drzewa indeksujące pozycje poszczególnych bajtów. Węzły indeksu zawierają pary  $(count[i], page\#[i])$ , gdzie **count** jest względną pozycją bajtu w poddrzewie o korzeniu **page**.



# Struktura dużych obiektów

Narzut na rozmiar dużych obiektów

Dla `size(count) = 4B` i `size(page) = 4B`

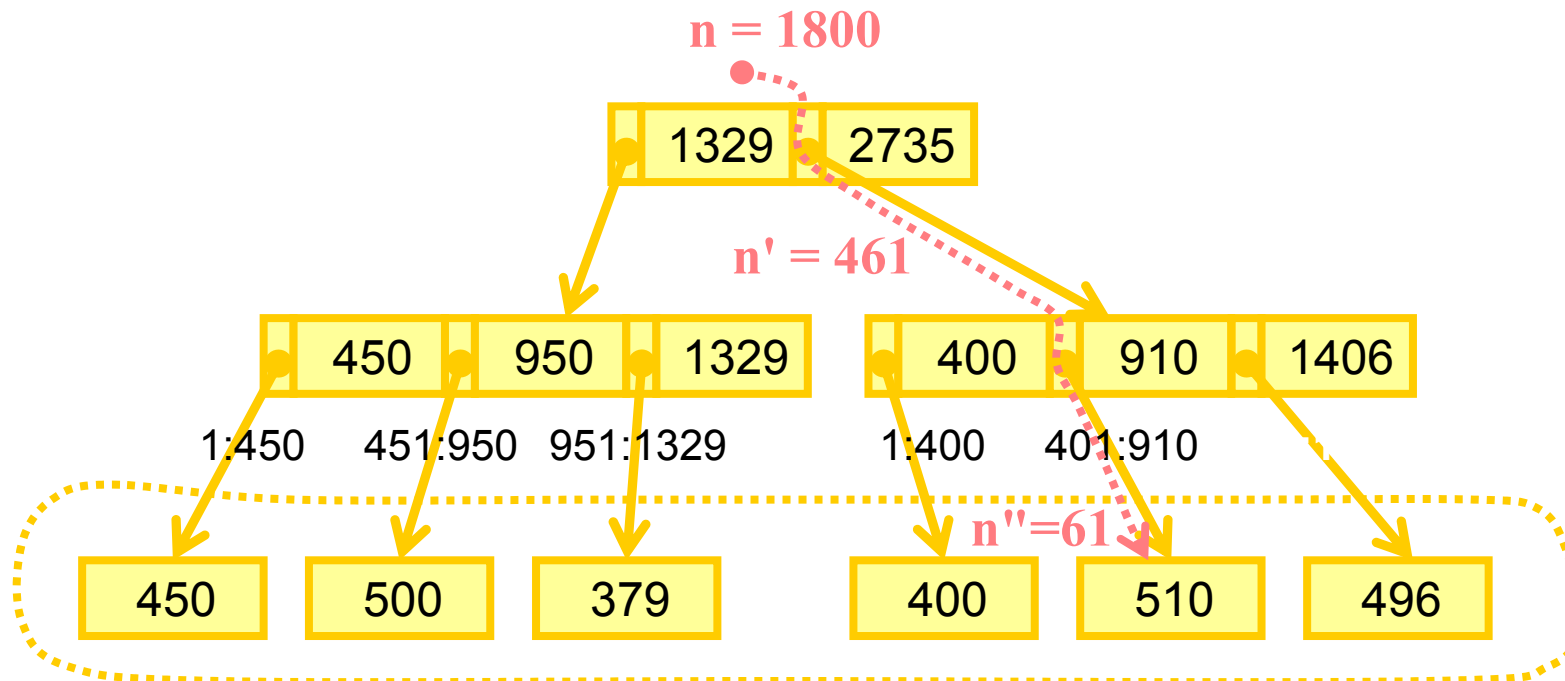
Liczba poziomów indeksu	Rozmiar bloku	Rozmiar obiektu	Rozmiar indeksu	Narzut indeksu
1	1kB 4kB	2kB – 128kB 8kB - 2MB	1kB 4kB	0,8-50% 0,2-50%
2	1kB 4kB	128kB – 16MB 2MB - 1GB	3kB-128kB 12kB-2MB	0,8-2% 0,2-0,6%
3	1kB 4kB	8MB-2GB 512MB-0,5TB	129kB-16MB 2MB-1GB	0,8-1,6% 0,2-0,4%
4	1kB 4kB	512MB-256GB 128GB-256TB	8,1MB-2G 0,5GB-0,5TB	0,8-1,6% 0,2-0,4%

# Wyszukiwanie pod-sekwencji bajtów

`Find(LOB o, int długość, int pozycja)`

Na przykład:

`Find(o1, 100, 1800)`



# Algorytm wyszukiwania pod-sekwencji bajtów

begin

start := S;

P := root page;

while (P != leaf page)

begin

znajdź min(count[i]) takie, że start ≤ count[i];

start := start - count[i-1]; //przeliczenie pozycji

/\* z definicji count[0]=0 \*/

push(P, i) on ST; //dla umożliwienia odczytu kolej. liści

P := read page#[i];

end;

string = P[start: min(start + N, count(P))];

N := N - min(N, count(P) - start);

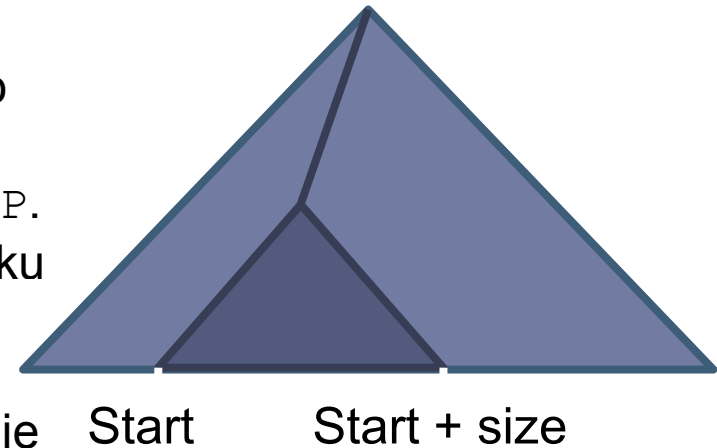
/\* jeżeli podsekwencja nie mieści się w pojedynczym bloku –  
odczytaj kolejne liście \*/

end

# Algorytm wstawiania do dużego obiektu

**Insert** (BLOB *o*, int *Start*, int *Size*,  
BLOB *s*)

1. Poszukaj strony dyskowej *P* zawierającej bajt *o* pozycji *Start*
2. Wstaw *Size* bajtów podsekwencji *s* do strony *P*. Jeżeli nie wystąpi przepełnienie przejdź do kroku 5.
3. Jeżeli wystąpi przepełnienie sprawdź czy nadmiarowe bajty podsekwencji *s* zmieszczą się u sąsiadów strony *P*. Jeżeli tak rozprosz równomiernie bajty *s* między stronę *P* i jej sąsiadów.
4. Jeżeli nie utwórz  $\lfloor \text{Size} / \text{rozmiar bloku} \rfloor$  nowych stron dyskowych. Rozprosz równomiernie wszystkie *Size* bajtów między stroną *P* i nowo utworzone stronami.
5. Propaguj rozmiary i identyfikatory zmodyfikowanych lub nowo utworzonych stron dyskowych do węzłów pośrednich indeksu

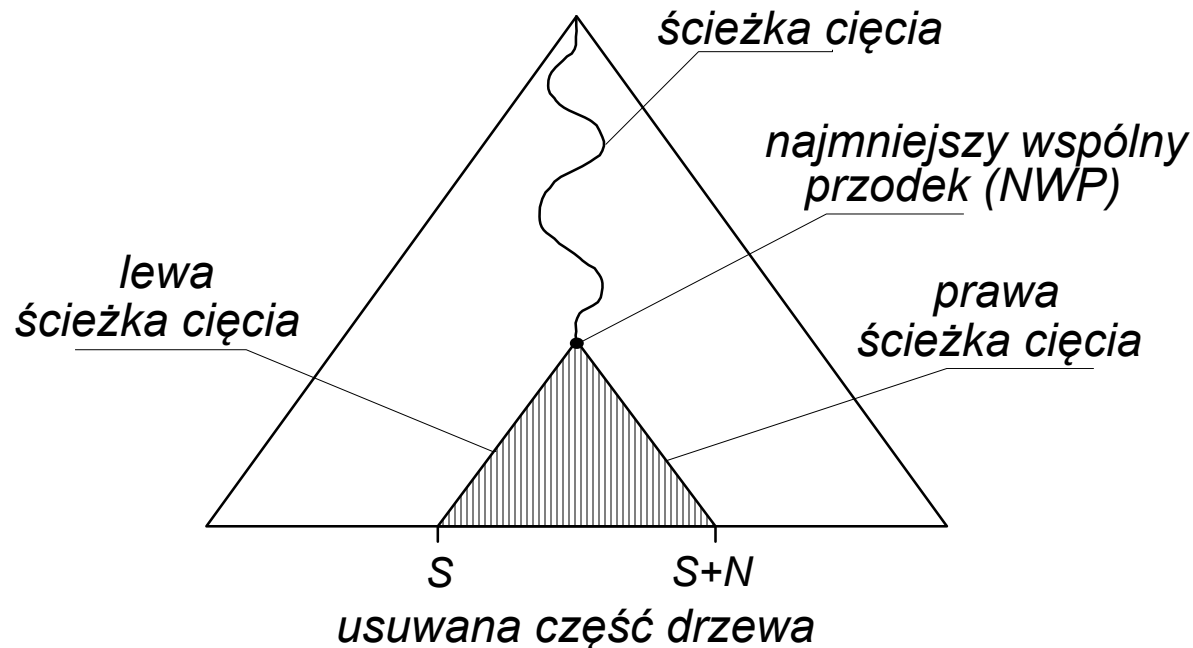


# Algorytm usuwania pod-sekwencji w dużym obiekcie

**Delete** (usuń  $N$  bajtów, z obiektu  $o$ , począwszy od pozycji  $S$ )

## Algorytm dwu-fazowy

1. Faza usuwania
2. Faza wyrównywania





# Węzły zagrożone

Węzeł jest *niepełny* jeżeli spełnia jeden z poniższych warunków:

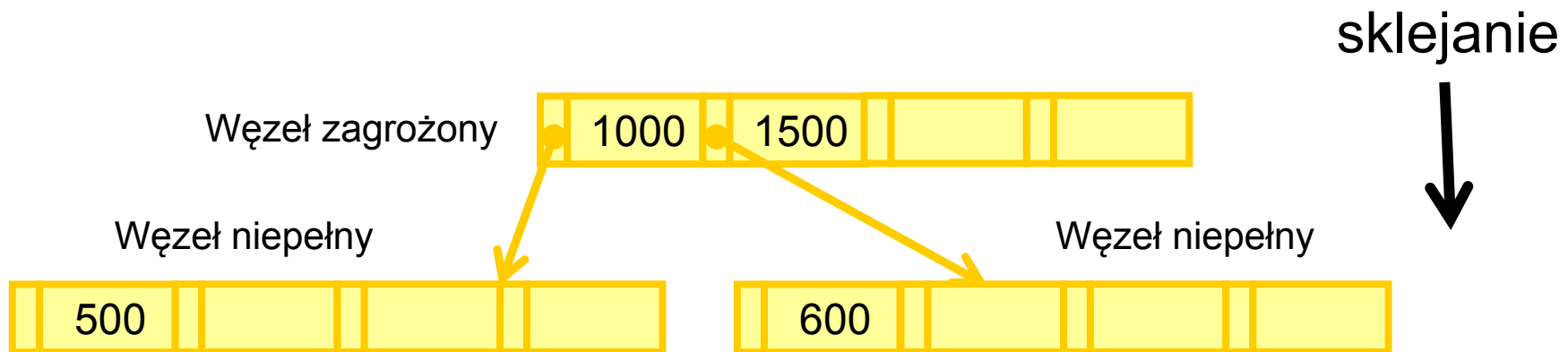
1. węzeł jest liściem i jest wypełniony mniej niż w połowie;
2. węzeł jest węzłem pośrednim i jest wypełniony mniej niż w połowie (dla korzenia  $< 2$  elementy).

Węzeł jest *zagrożony* jeżeli spełnia jeden z poniższych warunków:

1. węzeł jest *niepełny*;
2. węzeł jest węzłem pośrednim i jego wypełnienie jest równe  $p/2$  (dla korzenia  $= 2$ ) i jeden z jego bezpośrednich potomków leżących na ścieżce cięcia jest *zagrożony*;
3. węzeł jest węzłem NWP i jego wypełnienie jest równe  $p/2 + 1$  (dla korzenia 3) i obydwaj potomkowie leżący na ścieżce cięcia są zagrożeni.

# Węzły zagrożone

- ▶ Celem ustalenia węzłów zagrożonych jest zapewnienie jednoprzebiegowości fazy sklejanego drzewa.



# **Algorytm operacji Delete**

***Delete (N bajtów, od pozycji S, w obiekcie o)***

## ***Faza usuwania:***

1. wyznacz *lewą ścieżkę cięcia* (search (S));
2. wyznacz *prawą ścieżkę cięcia* (search (S+N));
3. usuń wszystkie węzły znajdujące się w całości między ścieżkami cięcia oraz uaktualnij liczniki wszystkich węzłów znajdujących się na ścieżkach cięcia;
4. wśród węzłów znajdujących się na ścieżkach cięcia zaznacz węzły *zagrożone*.

# Algorytm operacji Delete

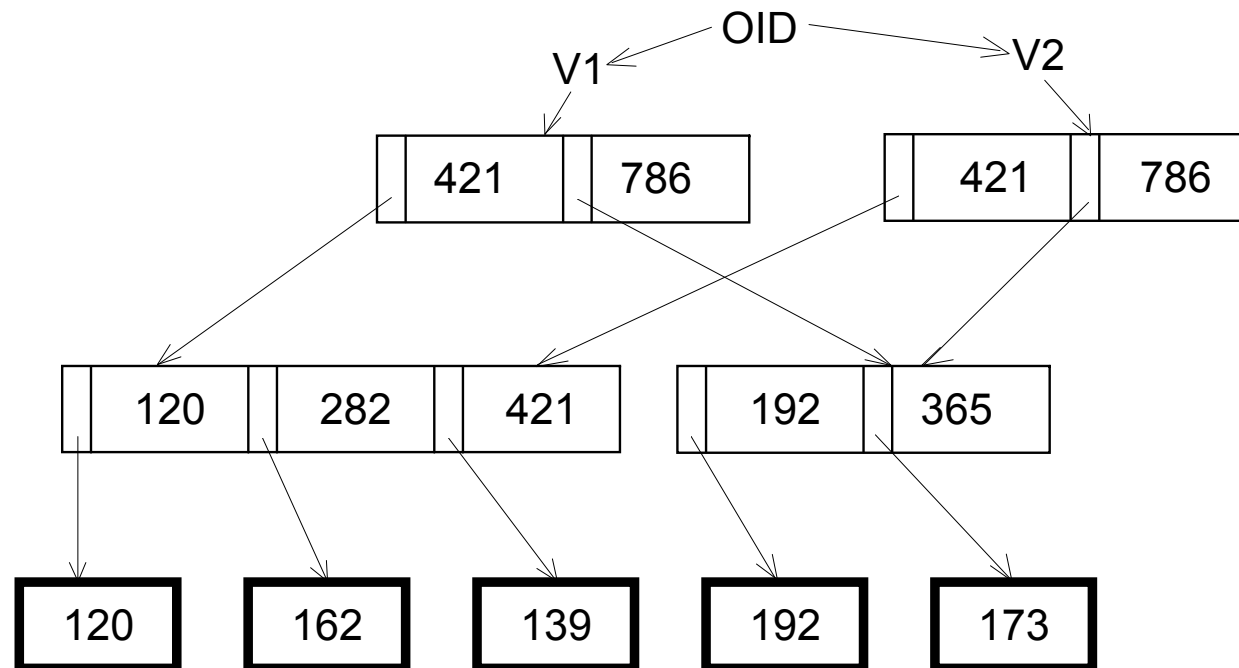
## ***Faza wyrównania drzewa:***

1. jeżeli korzeń nie jest węzłem *zagrożonym*, przejdź do kroku (2); jeżeli korzeń ma tylko jednego następcę usuń korzeń i przejdź do kroku (1), w przeciwnym przypadku połącz potomka zagrożonego z innym sąsiednim potomkiem lub przesunij bajty między sąsiednimi potomkami.
2. trawersuj wzdłuż ścieżki cięcia; po dojściu do ostatniego węzła drzewo jest zbalansowane;
3. tak długo jak bieżący węzeł jest *zagrożony*, wykonuj operację połączenia go z węzłem sąsiednim lub przesunięcia z sąsiedniego – niezagrożonego węzła odpowiedniej liczby elementów;
4. przejdź do kroku (2).



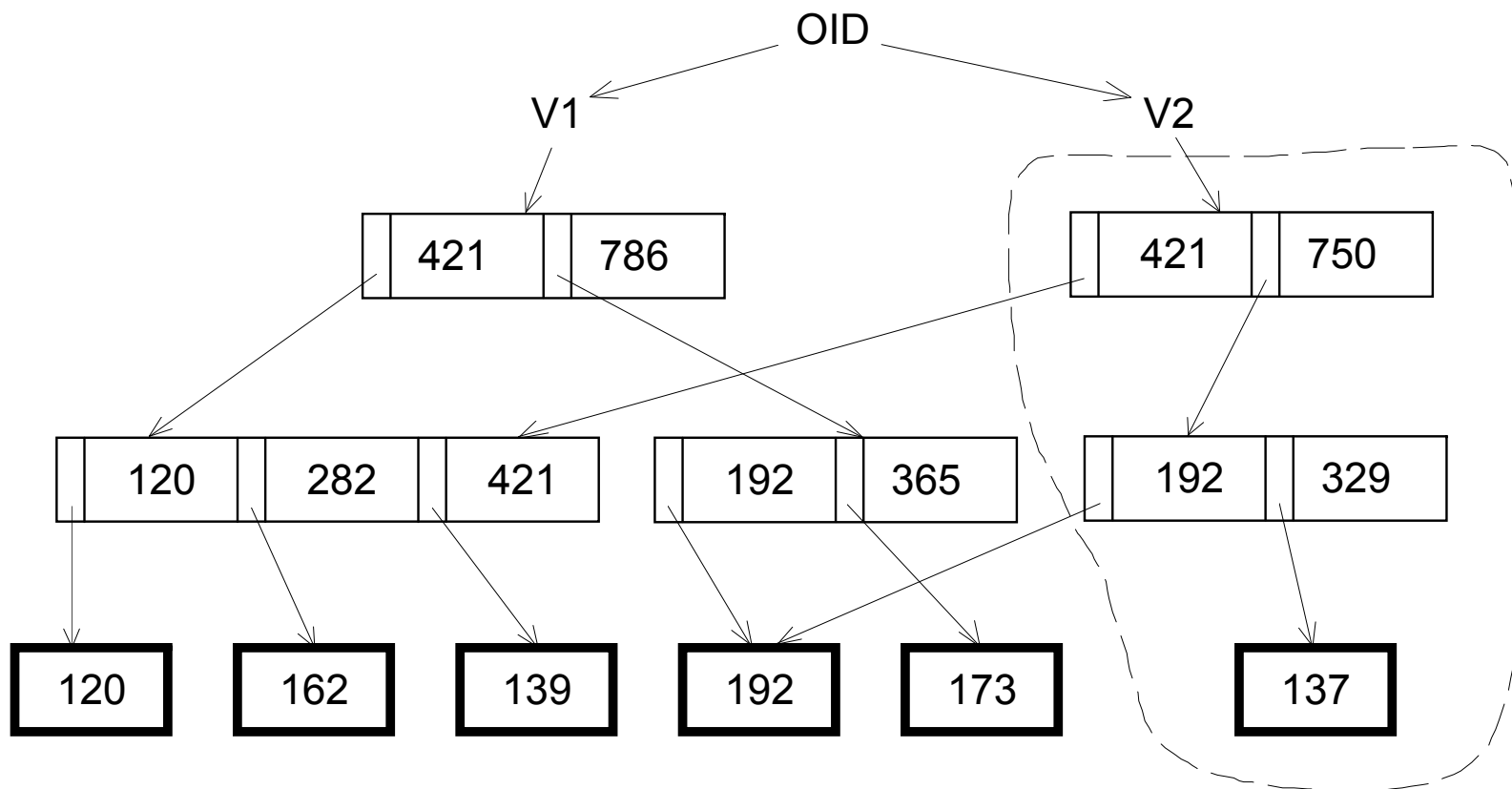
# Wersjowanie dużych obiektów

Założenie - wersje obiektów różnią się jedynie w niewielkim stopniu



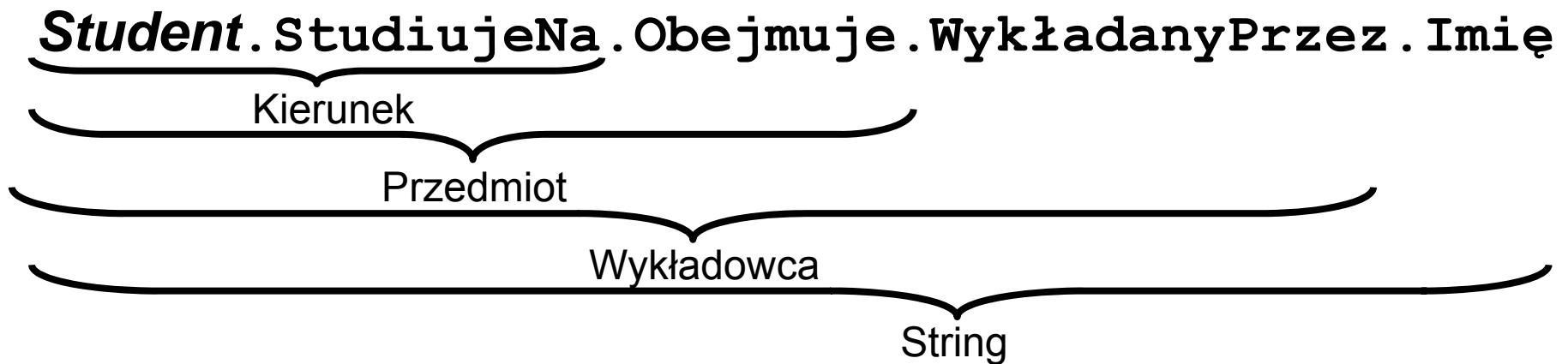
# Wersjowanie dużych obiektów

*v2.delete(36, 751)* - zmodyfikuj drugą wersję obiektu



# Indeksowanie wyrażeń ścieżkowych

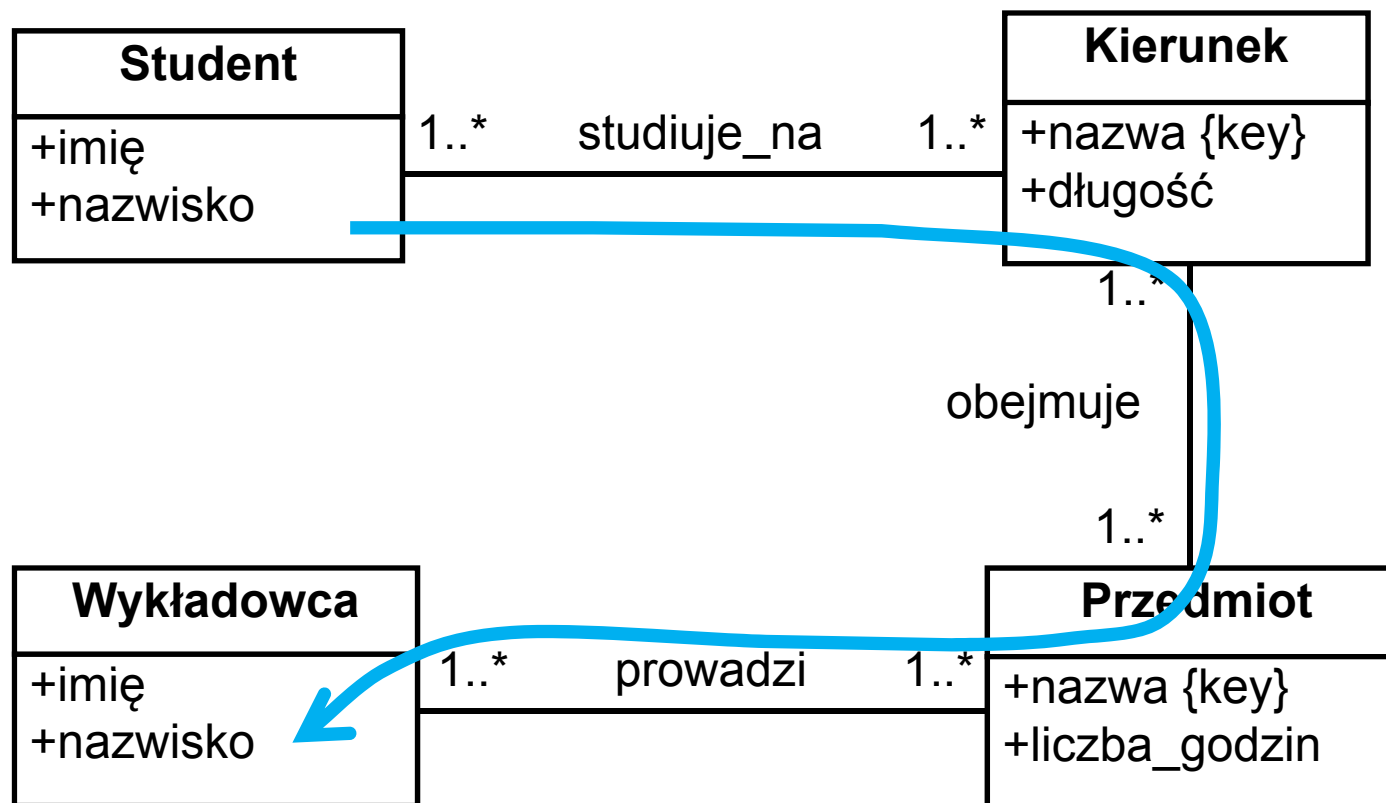
*Znajdź imiona wykładowców, wykładających przedmioty przypisane do kierunku, na którym studiuje dany student.*



**W celu przyśpieszenia nawigacji wzdłuż długich ścieżek:**

- ▶ Materializacja kompletnych ścieżek – relacje ASR
- ▶ Hierarchia kombinacji ścieżek binarnych

# Wyrażenia ścieżkowe - przykład





# Relacje ASR

## Dana baza obiektów:

<id1,'Józef','Nowak',21, {id2, id3}> :	Student
<id2,'Informatyka', 7, {id1}, {id4, id5}> :	Kierunek
<id3,'Zarządzanie', 7, {id1}, { }> :	Kierunek
<id4,'Bazy danych', 45, {id2}, {id6}> :	Przedmiot
<id5,'Grafika', 30, {id2}, {id7}> :	Przedmiot
<id6,'Jan', 'Tarzan', {id4}> :	Wykładowca
<id7,'Henryk', 'Nowek', {id5}> :	Wykładowca
<id8,'Józef', 'Tarzan', { }> :	Wykładowca

## Relacja ASR:

S <sub>0</sub> :OID <sub>student</sub>	S <sub>1</sub> :OID <sub>kierunek</sub>	S <sub>2</sub> :OID <sub>przedmiot</sub>	S <sub>3</sub> :OID <sub>wykładowca</sub>	S <sub>4</sub> :string
id1	id2	id4	id6	'Tarzan'
id1	id2	id5	id7	'Nowek'

# Klasy rozszerzeń ASR

Relacje ASR pozwalają na materializację całych ścieżek dla najczęściej zadawanych zapytań. Wyróżnia się cztery typy rozszerzeń relacji ASR.

- ▶ **Kanoniczne** – zawiera jedynie kompletne ścieżki, rozpoczynające się w  $t_0$  i kończące w  $t_n$ . (Tabela na poprzedniej stronie.)
- ▶ **Lewostronne** – zawiera wszystkie ścieżki zaczynające się w  $t_0$ , które jednak nie muszą się kończyć w  $t_n$ .

$S_0:OID_{student}$	$S_1:OID_{kierunek}$	$S_2:OID_{przedmiot}$	$S_3:OID_{wykładowca}$	$S_4:string$
id1	id3			

# Klasy rozszerzeń ASR

**Prawostronne** - zawiera wszystkie ścieżki, które kończą się w  $t_n$ , które jednak nie muszą się rozpoczynać w  $t_0$ .

$S_0:OID_{student}$	$S_1:OID_{kierunek}$	$S_2:OID_{przedmiot}$	$S_3:OID_{wykładowca}$	$S_4:string$
			id8	'Tarzan'

**Kompletne** – zawiera wszystkie kompletne i wszystkie częściowe ścieżki, które nie muszą się zaczynać w  $t_0$  i nie muszą kończyć w  $t_n$ .

$S_0:OID_{student}$	$S_1:OID_{kierunek}$	$S_2:OID_{przedmiot}$	$S_3:OID_{wykładowca}$	$S_4:string$
id1	id2	id4	id6	'Tarzan'
id1	id2	id5	id7	'Nowek'
id1	id3			
			id8	'Tarzan'

# Stosowalność relacji ASR

Dana relacja ASR( $S_0:OID_{t_0}, \dots, S_n:OID_{t_n}$ ) zawierająca rozszerzenie  $X$  jest **stosowalna** dla zapytania zawierającego ścieżkę  $s.A_i \dots A_j$  gdzie  $s$  jest wystąpieniem typu (lub podtypu)  $t_{i-1}$  pod następującymi warunkami:

$$X_{\text{kompletne}} \wedge 1 \leq i \leq j \leq n$$

$$X_{\text{lewostronne}} \wedge 1 = i \leq j \leq n$$

$$X_{\text{prawostronne}} \wedge 1 \leq i \leq j = n$$

$$X_{\text{kanoniczne}} \wedge 1 = i \leq j = n$$

# Organizacja fizyczna relacji ASR

Dla przyspieszenia dostępu relacje ASR są składowane w dwóch redundantnych strukturach fizycznych. Osobno dla pierwszego i osobno dla ostatniego atrybutu relacji są utworzone szybkie struktury dostępu: indeksy B\*-drzewa lub pliki haszowe. Struktura plików haszowych jest stosowana domyślnie w wypadku, gdy wartościami atrybutu są identyfikatory obiektów. Rzadkie indeksy B\*-drzewo są zakładane w wypadku, gdy wartościami atrybutu są wartości proste.

Struktura uporządkowana ze względu na pierwszy atrybut relacji umożliwi wydajne wykonanie zapytania: *znajdź nazwisko wykładowcy studenta o oid równym id1*. Struktura uporządkowana ze względu na ostatni atrybut relacji umożliwi wydajne wykonanie zapytania: *znajdź kierunki, na których wyklada Tarzan*.



# Hierarchia ścieżek binarnych

Modyfikacja relacji ASR przez pominięcie pośrednich obiektów w wyrażeniach ścieżkowych oraz usunięcie redundantnych ścieżek.

Dana relacja ASR z rozszerzeniem kanonicznym:

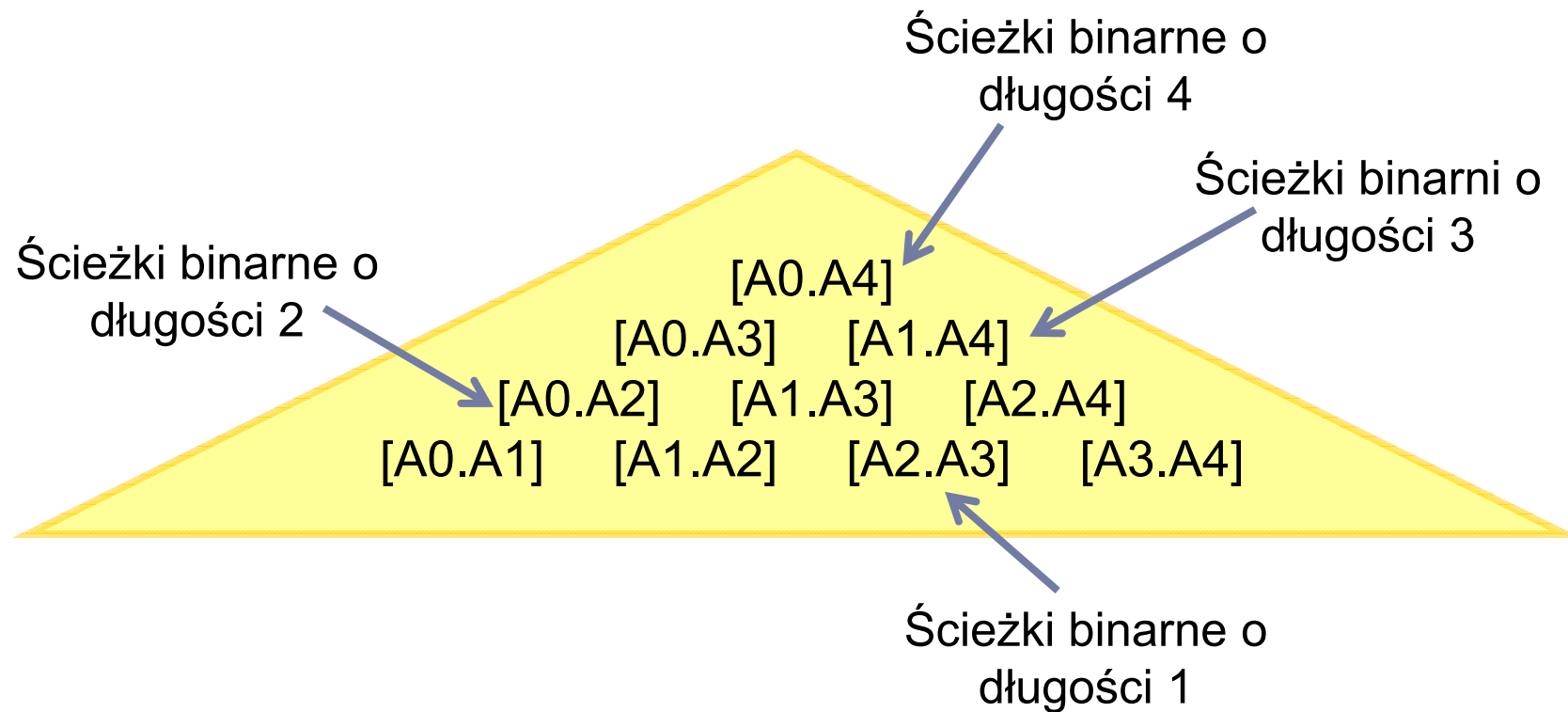
$S_0$ :OID <sub>student</sub>	$S_1$ :OID <sub>kierunek</sub>	$S_2$ :OID <sub>przedmiot</sub>	$S_3$ :OID <sub>wykładowca</sub>	$S_4$ :string
id1	id2	id4	id6	'Tarzan'
id1	id2	id5	id7	'Nowek'
id1	id3	id9	id8	'Tarzan'

Równoważne jej ścieżki binarne:

$S_0$ :OID <sub>student</sub>	$S_4$ :string	Count
id1	'Tarzan'	2
id1	'Nowek'	1

# Hierarchia ścieżek binarnych

Kompletna lub niekompletna hierarchia kombinacji ścieżek binarnych umożliwia wsparcie dla szerokiej klasy zapytań zawierających różne fragmenty ścieżek.



# Indeksowanie wyników metod

Materializacja i indeksowanie wyników metod ma na celu wydajne przetwarzanie zapytań odwołujących się do wyników metod:

```
select nr_indeksu, nazwisko  
from studenci  
where średnia_ocen( ) > 4.25
```

Struktura **GMR (Generalized Materialization Relation)** pozwala materializować wyniki dla ogólnego przypadku:

- ▶ wiele metod skojarzonych z pojedynczą klasą,
- ▶ metody z niepustym zbiorem parametrów wejściowych (nie licząc identyfikatora odbiorcy komunikatu).



# Struktura GMR

$\langle A_1:t_1, \dots, A_n:t_n, \quad m_1:t_{n+1}, \dots, m_k:t_{n+k}, V_1:bool, \dots V_k:bool \rangle$

Parametry  
wejściowe

Wyniki

Ważność

- ograniczona dziedzina
- typowe wartości
- historia wartości

**Ograniczona struktura GMR** – parametry  $A_1, \dots, A_n$  są zdefiniowane na klasach, których rozszerzenia są składowane w bazie danych.

A1 : Student	średnia_ocen : float	V <sub>średnia_ocen</sub> : bool
id1	2.13	True
id12	3.79	True
id13	4.89	False
...		

# Struktury GMR

## Szybkie ścieżki dostępu do GMR:

Indeksy wielowymiarowe na atrybutach:  $A_1, \dots, A_n, m_1, \dots, m_k$ .

Utrzymywanie struktury GMR

- ▶ Poprawność GMR
- ▶ Kompletność GMR

## Dwie podstawowe strategie utrzymania poprawności:

- ▶ **Natychmiastowa re-materializacja** – natychmiast po zmianie wartości obiektów, dla których są wywoływane metody
- ▶ **Opóźniona re-materializacja** – składowane wyniki metod, które po modyfikacji stanu bazy danych stały się niepoprawne, są zaznaczane jako nieważne przez ustawienie wartości False odpowiedniego atrybutu  $V_i$

## Wydajność struktur GMR

Dla wydajnego utrzymywania aktualnego stanu struktury GMR, jest niezbędne szybkie znalezienie wszystkich krotek struktury GMR, które staną się niepoprawne w wyniku modyfikacji obiektu **o**. W tym celu jest stosowana dodatkowa struktura referencji zwrotnych **RRR** (Reverse Reference Relation).

Struktura RRR:

**<o:OID, f:FunctionID, A:<OID>>**

gdzie: o – jest obiektem, którego wartość wpływa na wynik indeksowanej metody

f – jest indeksowaną metodą

A – jest obiektem, na rzecz którego metoda jest wykonywana

# Wydajność struktur GMR

Baza danych:

<id1, 'Józef', 'Nowak', 21, {id6, id7}>:Student

<id2, 'Henryk', 'Nowek', {id8}>:Student

<id3, 'Józef', 'Buła', { }>:Student

<id4, 'Bazy danych', 45, ...>:Przedmiot

<id5, 'Grafika', 30, {id2}, {id7}>:Przedmiot

<id6, {id4}, 4>:Egzamin

<id7, {id5}, 5>:Egzamin

<id8, {id4}, 2>:Egzamin

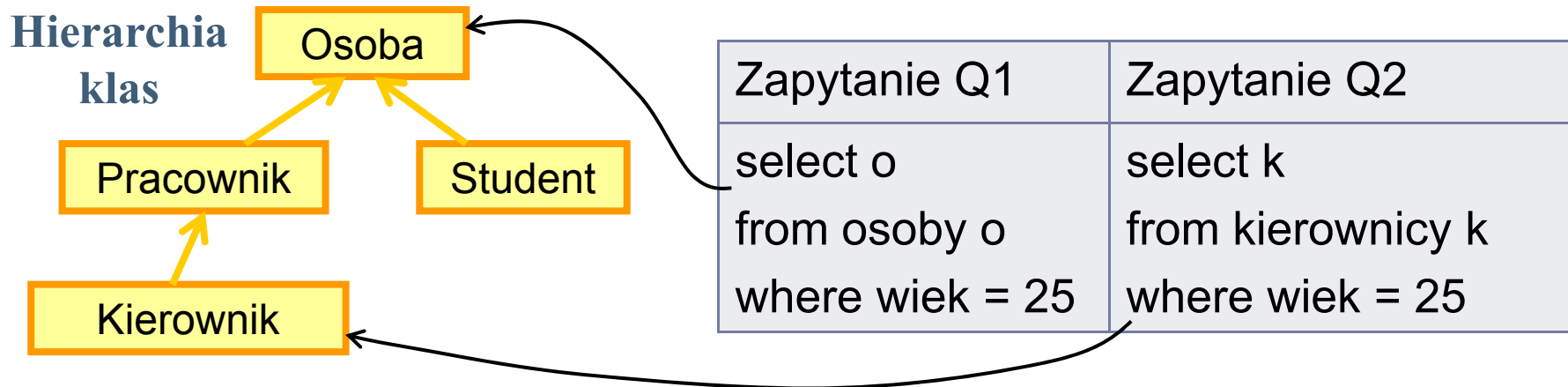
<b>o</b>	<b>f</b>	<b>A</b>
id6	<b>Student.średnia_ocen</b>	id1
id7	<b>Student.średnia_ocen</b>	id1
...		

# Wydajność struktur GMR

## Techniki optymalizacji indeksowania metod:

1. Utrzymywanie informacji o atrybutach obiektów mających wpływ na wyniki metod. Wtedy modyfikacja nieistotnych cech nie będzie wyzwała operacji weryfikacji potrzeby re-materializacji
2. Utrzymywanie bezpośrednio w obiektach informacji o występowaniu w strukturze RRR
3. Wykorzystanie hermetyczności, dla ograniczenia modyfikacji do publicznych metod
4. Kompensacja składowanych wyników metod zamiast ich pełnego przeliczania

# Indeksowanie hierarchii rozszerzeń klas



Zastosowanie klasycznych indeksów:

- ▶ Jeden indeks dla wszystkich podzbiorów w hierarchii - wydajna realizacja zapytań klasy Q1, niewydajna zapytań klasy Q2.
- ▶ Osobne indeksy dla każdego podzbioru – wydajna realizacja zapytań klasy Q2, niewydajna zapytań klasy Q1.

# Indeksowanie hierarchii rozszerzeń klas

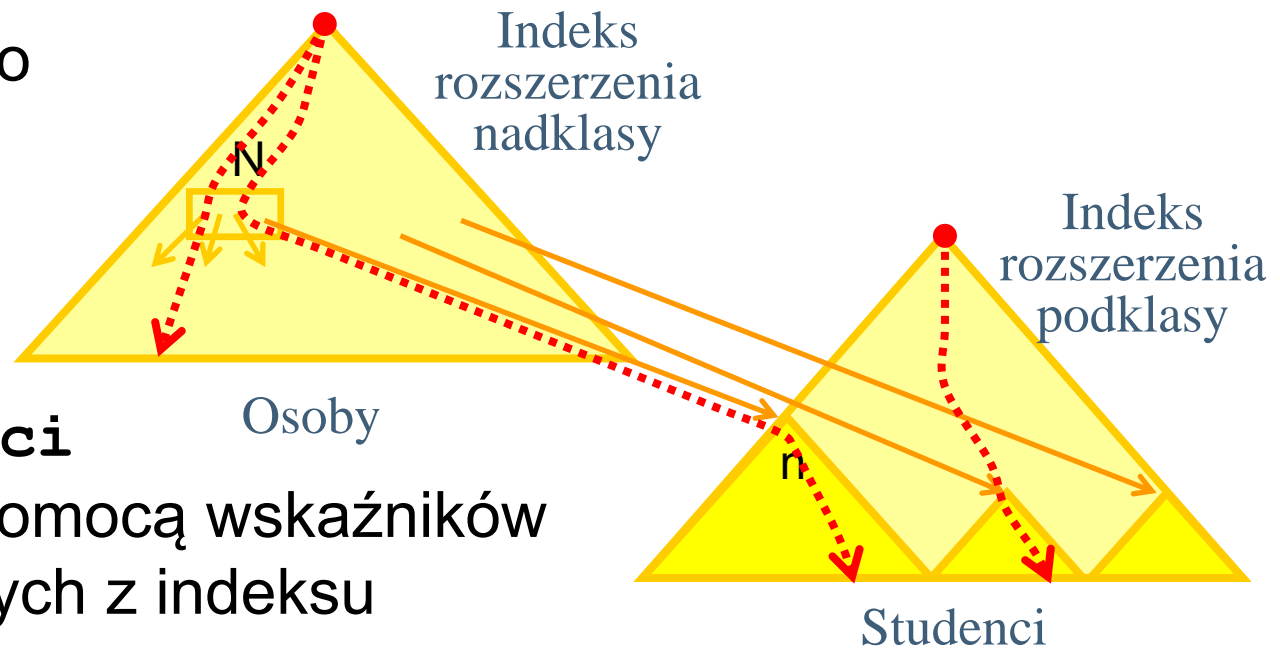
## Nowe wyspecjalizowane rodzaje indeksów:

## ► Indeks hierarchii klas

...	klucz =25	Pracownicy:{ $p_{11}$ , $p_{12}$ , ...}	Studenci :{ $p_{21}$ , $p_{22}$ ..., }	...	klucz =26	...
-----	-----------	---	--	-----	-----------	-----

- Indeks h-drzewo

Liście indeksu podklasy **Studenci** są osiągalne za pomocą wskaźników między-indeksowych z indeksu nadklasy **Osoby**



# Związki wielokrotne i atrybuty wielowartościowe

```
class Wielokąt {  
    attribute set<Punkt> wierzchołki; }  
class Koło {  
    attribute Punkt środek;  
    attribute Float promień; }
```

Znajdź dla studentów przedmioty, na które mogą się zapisać ze względu na już zaliczone przedmioty:

```
select p, s  
from p in przedmioty join s in studenci  
on p.wymaga_ukończenia submultiset s.zaliczył
```

Powyższego zapytania nie można wykonać za pomocą szybkich algorytmów: sort-merge lub hash-join.



# Sygnatury zbiorów

- ▶ Każdy element zbioru jest odwzorowany przez unikalną sygnaturę
- ▶ Za pomocą sumowania logicznego sygnatury wszystkich elementów zbioru są składane w pojedynczą sygnaturę aproksymującą cały zbiór
- ▶ Dla operatora podzbioru sygnatury zbiorów muszą spełniać następującą zależność:

$$s \subseteq t \Rightarrow \text{sig}(s) \& \sim \text{sig}(t) = 0$$

Sig('Bazy Danych') → 00010001

Sig('Grafika komputerowa') → 10001000

Sig({'Bazy Danych', 'Grafika komputerowa'}) → 10011001

# Zastosowanie sygnatur zbiorów

- ▶ W algorytmie nested-loop dla ustalania relacji między atrybutami lub związkami wielowartościowymi. Generowanie i porównywanie sygnatur zbiorów zamiast samych zbiorów ogranicza złożoność obliczeniową algorytmu.
- ▶ W algorytmie sort-merge dla porządkowania zbiorów ze względu na wartości sygnatur;
- ▶ W algorytmie hash-join dla wyznaczania argumentu funkcji haszowej;
- ▶ Do indeksowania atrybutów wielowartościowych

# Generowanie sygnatur zbiorów

Dla operatorów porównania zbiorów innych niż  $=$  i  $\neq$ , niezbędne jest wygenerowanie sygnatur wszystkich zbiorów spełniających dany warunek. Na przykład, dla operatora podzbioru algorytm ten wygląda następująco:

```
s = a & -a;  
while(s) {  
    s = a & (s - a);  
    process(s);  
}
```

gdzie  $a$  jest sygnaturą zbioru wejściowego, a wartości  $s$  przebiegają po wszystkich sygnaturach potencjalnych podzbiorów zbioru wejściowego.

Na przykład dla zbioru, którego sygnatura  $a$  jest równa 1001 algorytm wygeneruje sygnatury:

1000, 0001, 1001, 0000

# Indeks RD-drzewo

Wartościami indeksowanymi przez indeks RD-drzewo są sygnatury związków wielokrotnych lub atrybutów wielowartościowych.

