Zwielokrotnianie i spójność

Zwielokrotnianie

Zwielokrotnianie polega na utrzymywaniu wielu kopii danych (obiektów) na niezależnych serwerach

Cele zwielokrotniania

1. zwiększenie efektywności
2. zwiększenie niezawodności (i dostępności)

Główne problem – spójność

- modele spójności
- protokoły spójności
- mobilność

Po co zwielokrotniać?

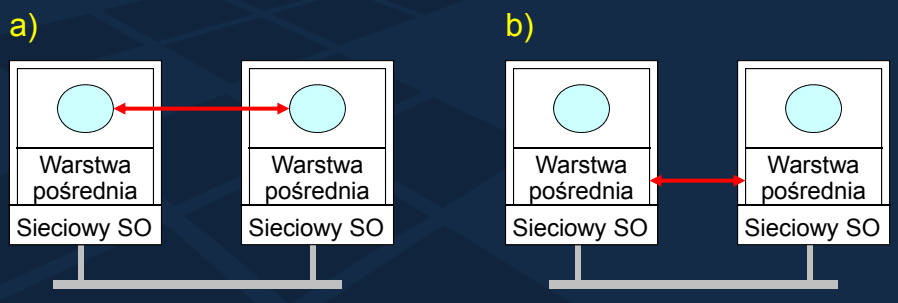
Niezawodność

- odporność na awarie

Efektywność

- współbieżny dostęp do wielu serwerów
  - równoważenie obciążenia
  - skalowalność w sensie liczbowym
- wykorzystanie bliższych serwerów
  - skalowalność w sensie geograficznym
  - mniejsze opóźnienia

Zwielokrotnianie obiektów



Zwielokrotnianie a skalowalność

Kompromis

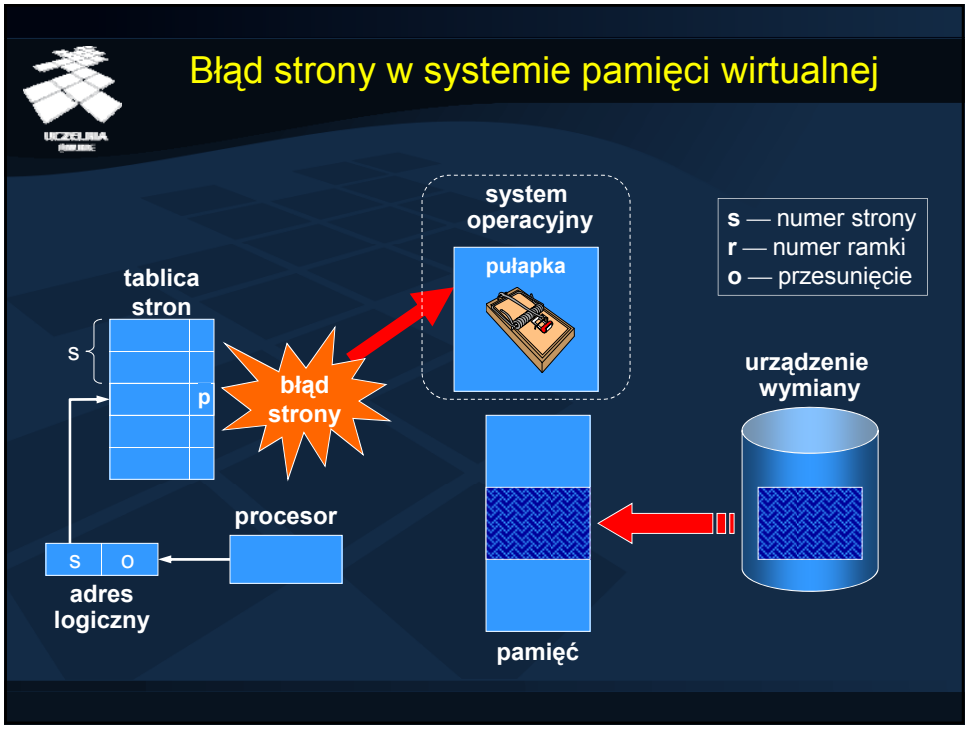
- skracanie czasu dostępu do serwerów
- utrzymywanie kopii w stanie spójnym
  - koszt utrzymywania nieużywanych kopii
- spójność odwzorowująca system scentralizowany
  - transakcyjna aktualizacja wszystkich kopii
  - globalna synchronizacja
- osłabienie modelu spójności
  - charakter aplikacji i zwielokrotnianych danych

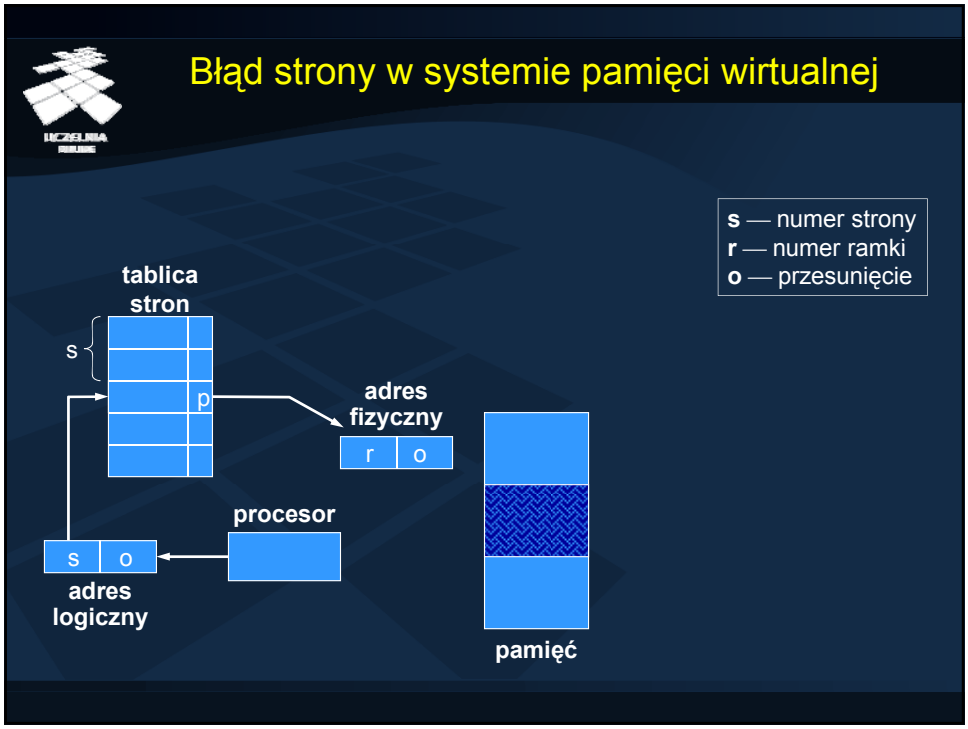
Rozproszona pamięć dzielona

DSM (ang. *Distributed Shared Memory*) — wspólna wirtualna przestrzeń adresowa, dostępna dla wszystkich węzłów systemu rozproszonego.

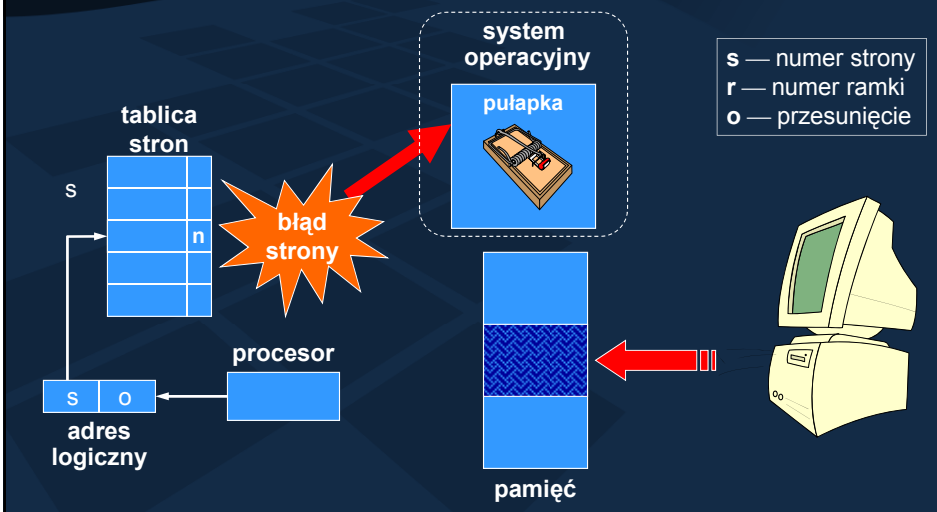
Zalety

- wygodny paradygmat programowania równoległego
- skalowalność i łatwość rozbudowy
- dostęp do fizycznej pamięci wszystkich węzłów
- środowisko uruchomieniowe dla programów równoległych pisanych dla maszyn wieloprocesorowych


Błąd strony w systemie pamięci wirtualnej
system operacyjny
pułapka
s — numer strony
r — numer ramki
o — przesunięcie
tablica stron
s
p
błąd strony
urządzenie wymiany
procesor
s
o
adres logiczny
pamięć

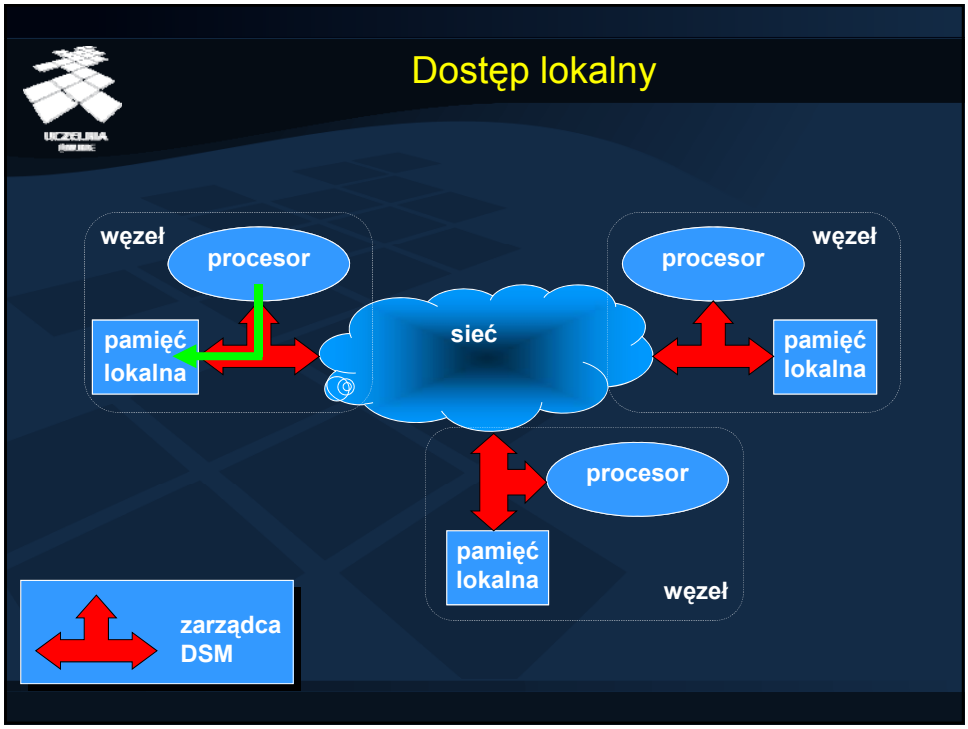

Błąd strony w systemie pamięci wirtualnej
s — numer strony
r — numer ramki
o — przesunięcie
tablica stron
s
p
adres fizyczny
r
o
procesor
s
o
adres logiczny
pamięć

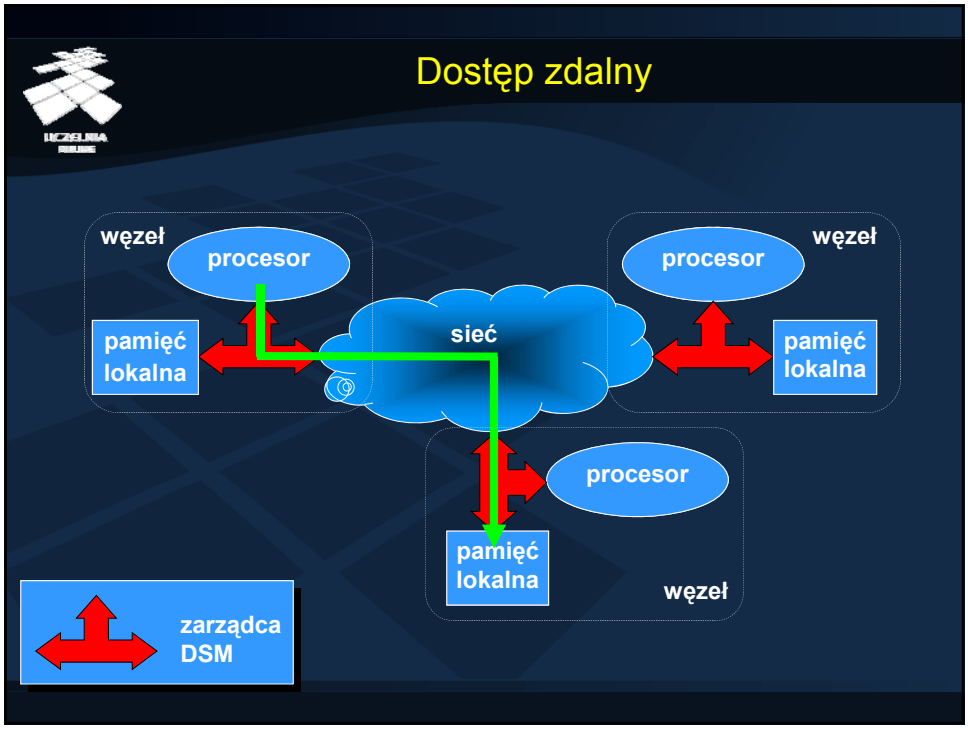
Obsługa błędu strony w systemie DSM

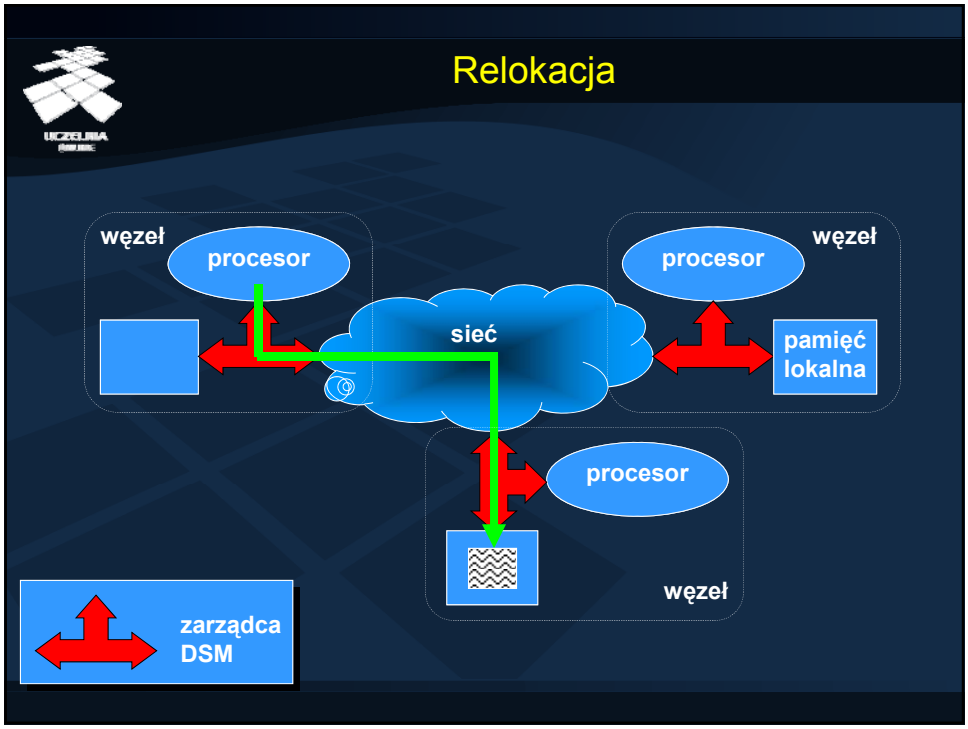


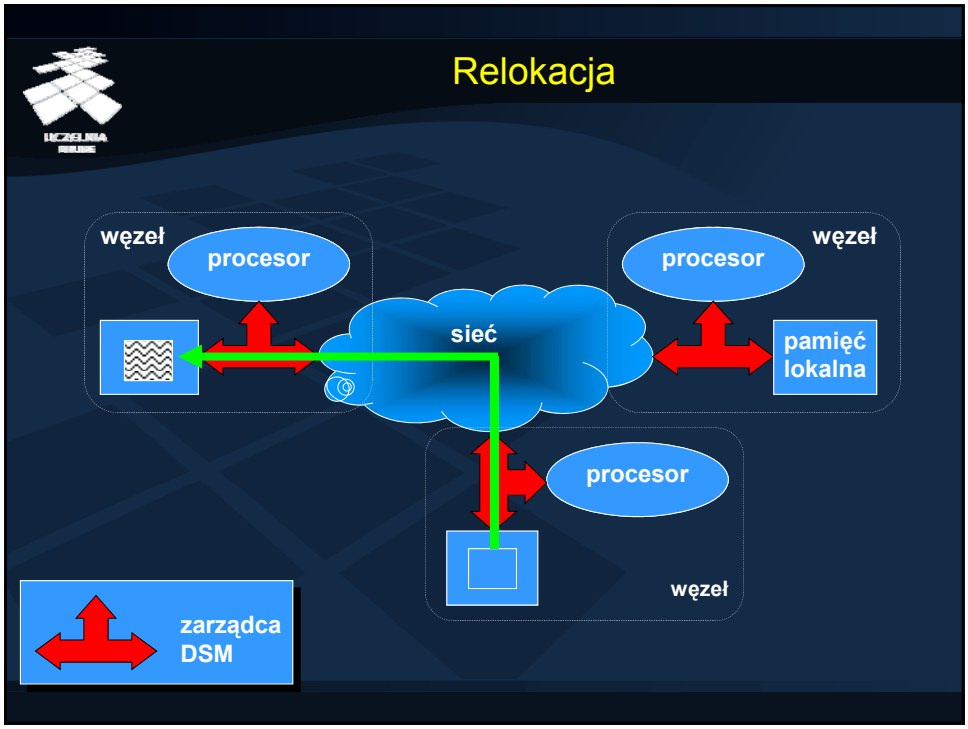
Koncepcje dostępu do danych

- Dostęp zdalny – zawsze poprzez sieć
  - prostota koncepcji i implementacji
  - opóźnienia komunikacyjne
- Relokacja – fizyczna zmiana lokalizacji obiektu
  - zmniejszenie czasu dostępu
  - koszt przenoszenia obiektu między węzłami
  - opłacalne przy wielokrotnych, zgrupowanych odwołaniach
- Zwielokrotnianie – kopie w lokalnych węzłach
  - zmniejszenie czasu dostępu
  - problem spójności

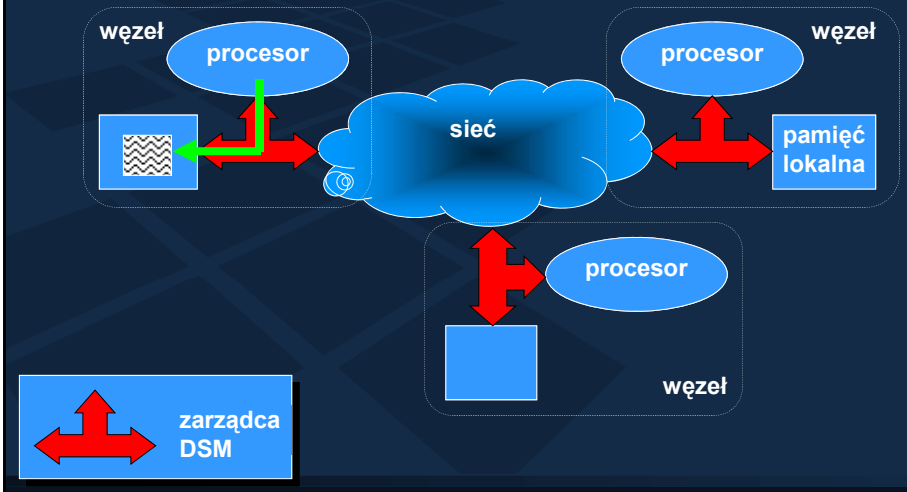

Dostęp lokalny
węzeł
procesor
pamięć lokalna
sieć
procesor
węzeł
pamięć lokalna
procesor
pamięć lokalna
węzeł
zarządca DSM


Dostęp zdalny
węzeł
procesor
pamięć lokalna
sieć
procesor
węzeł
pamięć lokalna
procesor
pamięć lokalna
węzeł
zarządca DSM


Relokacja
węzeł
procesor
procesor
węzeł
sieć
pamięć
lokalna
procesor
węzeł
zarządca
DSM

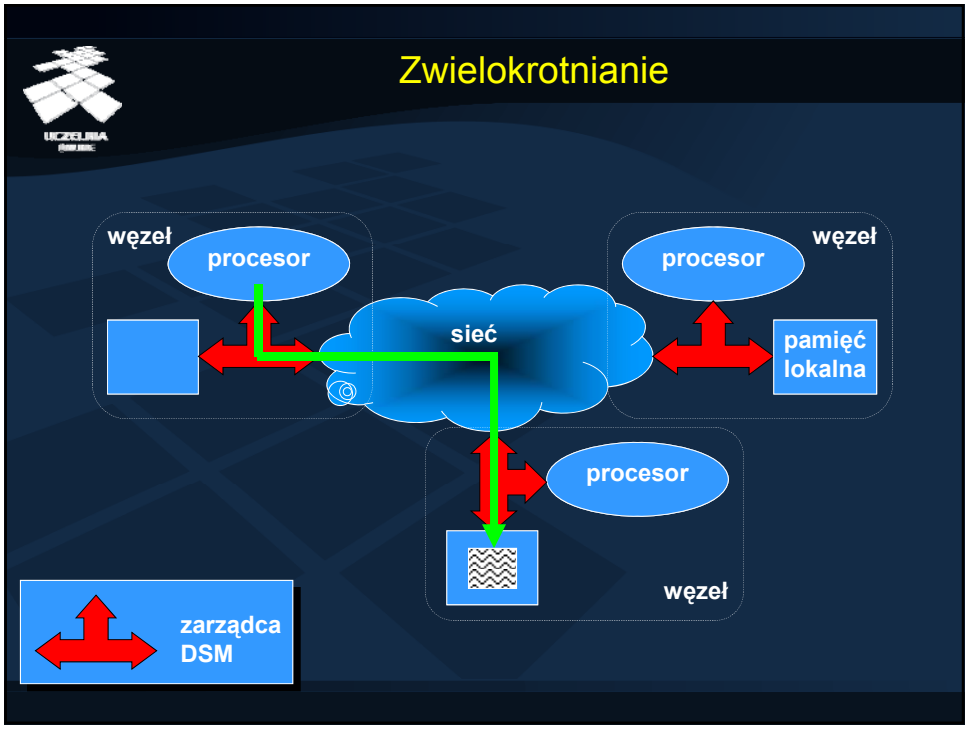

Relokacja
węzeł
procesor
procesor
węzeł
sieć
pamięć
lokalna
procesor
węzeł
zarządca
DSM

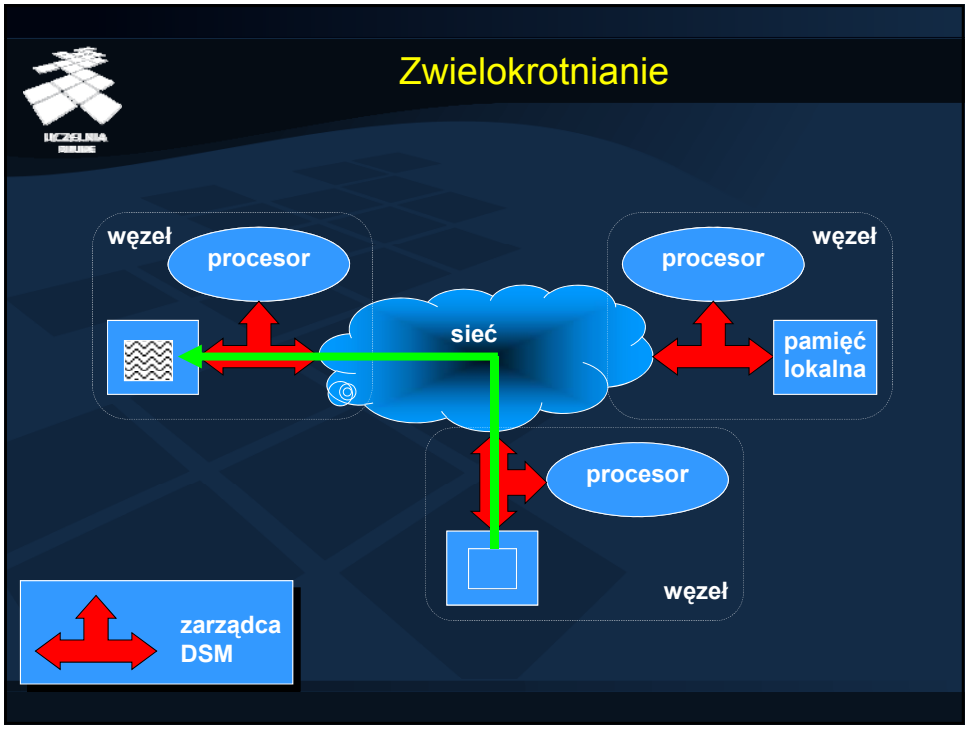
Relokacja



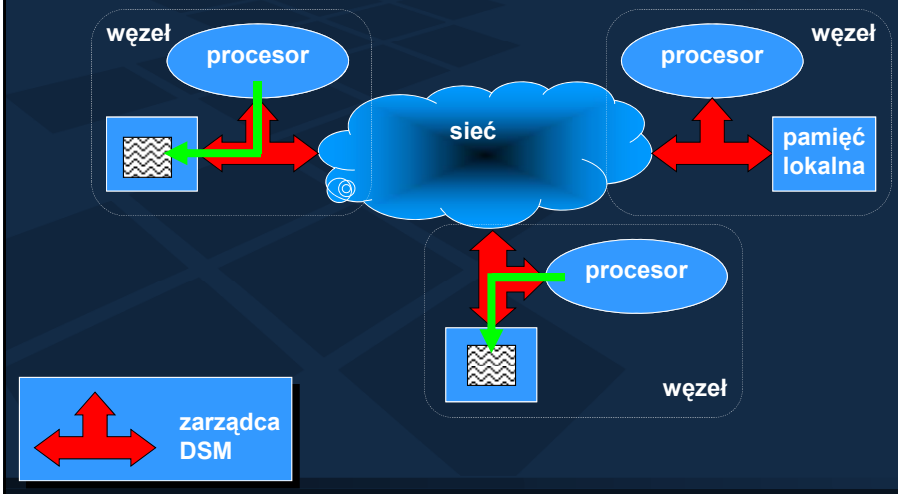
Relokacja — charakterystyka

1. Problem lokalizacji
   - adres obiektu zmienienia się w czasie
2. Problem rozmiaru i struktury przemieszczanej jednostki
   - małe obiekty → duży poziom współdzielenia
   - duże obiekty → mały narzut administracyjny
3. Problem migotania (ang. *trashing*, *ping-pong effect*)
   - naprzemienne odwołania kilku procesów


Zwielokrotnianie
węzeł
procesor
procesor
węzeł
sieć
pamięć
lokalna
procesor
węzeł
zarządca
DSM


Zwielokrotnianie
węzeł
procesor
procesor
węzeł
sieć
pamięć
lokalna
procesor
węzeł
zarządca
DSM

Zwielokrotnianie



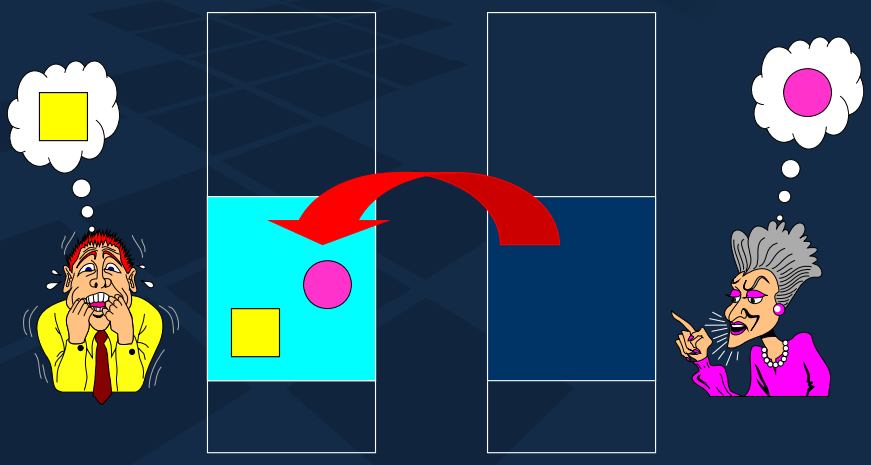
Zwielokrotnianie — charakterystyka

- Problem lokalizacji
  - tworzenie nowych replik
  - usuwanie starych replik
- Problem rozmiaru i struktury obiektów zwielokrotnianych
- Problem migotania nie występuje
  - kopia dla każdego ubiegającego się węzła
- Problem spójności kopii (replik)
  - stosunek liczby zapisów do odczytów

Struktura zwielokrotnianej jednostki

- **Strona** – fizyczne połączenie kilku odrębnych obiektów logicznych w jedną jednostkę udostępnianą jako całość przez **DSM** (*problem fałszywego współdzielenia*)
- **Pojedyncza zmienna** – duży jednostkowy koszt relokacji i utrzymywania spójności
- **Obiekt** (hermetyczna struktura danych udostępniana tylko przez zdefiniowane metody) – możliwość optymalizacji w strategii utrzymywania spójności w związku ze ściśle określonym sposobem dostępu (poprzez metody)

Fałszywe współdzielenie



Protokół koherencji

Protokół koherencji (spójności) jest algorytmem rozproszonym realizującym określony model spójności

1. Protokół **unieważniania** danych (ang. *invalidation protocol*)
   - małe komunikaty
   - jednokrotnie unieważnienie
2. Protokół **aktualizacji** danych (ang. *update protocol*) – niespójne repliki są aktualizowane
   - większe komunikaty

Model spójności

Model spójności określa gwarancje dotyczące spójności replik, dawane aplikacji (równoległej) przez system

- W jaki sposób definiować model spójności?
- W jaki sposób określić gwarancje dla aplikacji?
- Kiedy i w jaki sposób egzekwować te gwarancje?

Spójność ścisła

Spójność ścisła (ang. *strict consistency*)

- każdy odczyt zmiennej *x* zwraca wartość odpowiadającą wynikowi *ostatnio* wykonanej operacji zapisu

Systemy rozproszone

- niejednoznaczność określenia *ostatni* (brak globalnego zegara)
- duży koszt realizacji spójności ścisłej

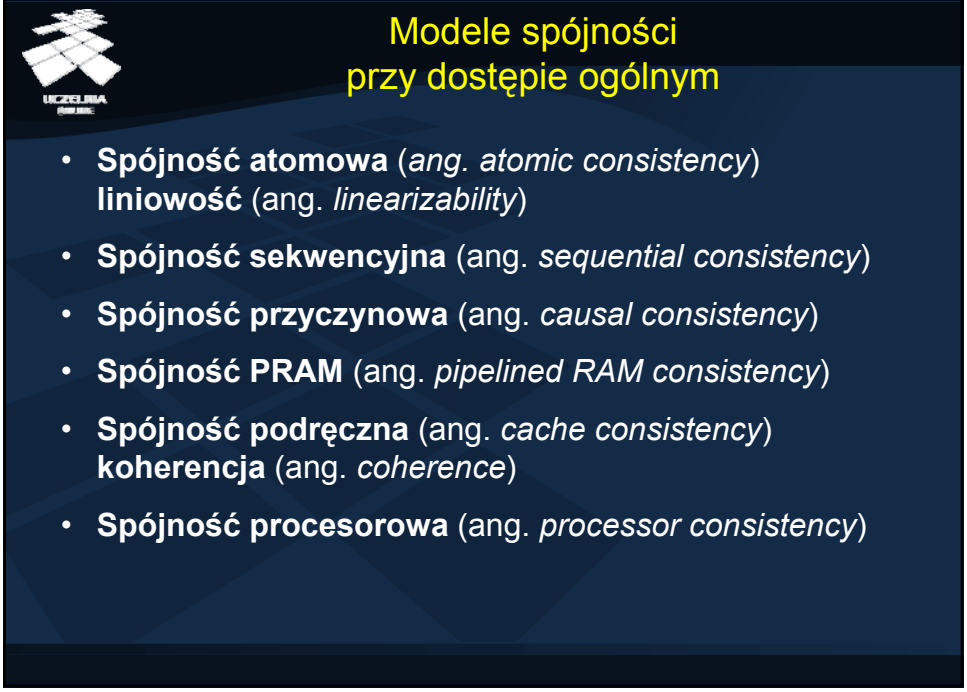
Modele spójności replik

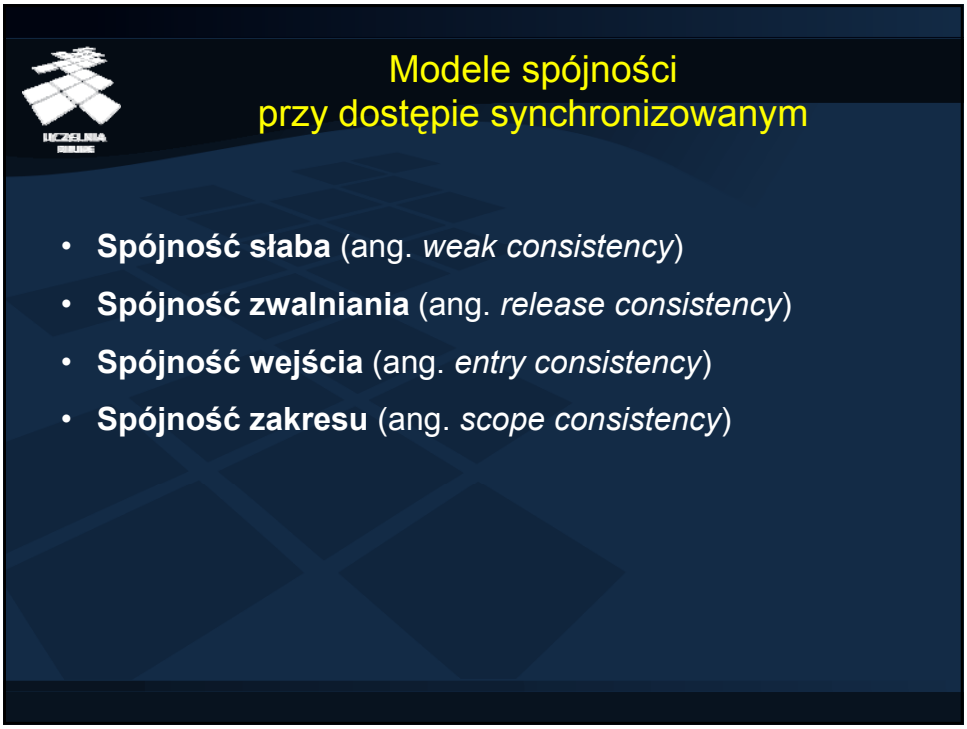
Modele spójności nastawione na dane

- modele spójności przy dostępie ogólnym
  uspójnianie danych przy każdej modyfikacji
- modele spójności przy dostępie synchronizowanym
  uspójnianie danych tylko podczas wykonywania jawnych operacji synchronizujących

Modele spójności nastawione na klienta

- uwzględnienie mobilności klienta


Modele spójności
przy dostępie ogólnym
• Spójność atomowa (ang. atomic consistency)
liniowość (ang. linearizability)
• Spójność sekwencyjna (ang. sequential consistency)
• Spójność przyczynowa (ang. causal consistency)
• Spójność PRAM (ang. pipelined RAM consistency)
• Spójność podręczna (ang. cache consistency)
koherencja (ang. coherence)
• Spójność procesorowa (ang. processor consistency)


Modele spójności
przy dostępie synchronizowanym
• Spójność słaba (ang. weak consistency)
• Spójność zwalniania (ang. release consistency)
• Spójność wejścia (ang. entry consistency)
• Spójność zakresu (ang. scope consistency)

Podstawowe założenia

- W skład systemu DSM wchodzą:
  - zbiór sekwencyjnych procesów $P = \{p_1, p_2, \ldots, p_n\}$
  - zbiór współdzielonych zmiennych $X = \{x_1, x_2, \ldots\}$
- Każdy proces ma własną replikę całego zbioru X
- Proces p_i może realizować na zmiennej $x \in X$ operacje:
  - zapisu wartości v, oznaczane jako $w_i(x)v$
  - odczytu wartości v, oznaczane jako $r_i(x)v$
- Realizacja operacji przebiega w dwóch fazach:
  - żądanie operacji (ang. *operation issue*)
  - wykonanie operacji (ang. *operation execution*)

Oznaczenia

$w_i(x)v$	zapis wartości v do zmiennej x wykonany przez proces p_i
$r_i(x)v$	odczyt wartości v ze zmiennej x wykonany przez proces p_i
O	zbiór wszystkich operacji w systemie
O_i	zbiór operacji procesu p_i (żądanych przez p_i)
OW	zbiór wszystkich operacji zapisu w systemie
$O\|x$	zbiór wszystkich operacji na zmiennej x
$\rightarrow_i$	lokalny porządek operacji procesu p_i
$\rightarrow$	porządek przyczynowy
$\mapsto_i$	uszeregowanie operacje postrzegane są przez proces p_i

Definicja porządku przyczynowego

(1) $\forall_{o1,o2 \in O_i} (o1 \rightarrow_i o2 \Rightarrow o1 \rightarrow o2)$

(2) $\forall_{x \in X} w(x)v \rightarrow r(x)v$

(3) $\forall_{o1,o2,v \in H} \left[(o1 \rightarrow o \wedge o \rightarrow o2) \Rightarrow o1 \rightarrow o2 \right]$

Definicja uszeregowania legalnego

- Uszeregowanie $\mapsto_i$ jest legalne $\Leftrightarrow$

$$\forall_{\substack{w(x)v \in OW \\ r(x)v \in O_i}} \left(\begin{array}{l} w(x)v \mapsto_i r(x)v \\ \wedge \quad \nexists_{o(x)u \in O_i \cup OW} \left[u \neq v \wedge w(x)v \mapsto_i o(x)u \mapsto_i r(x)v \right] \end{array} \right)$$

UWAGA
W celu uproszczenia **definicji** zakłada się, że każda operacja zapisu danej zmiennej zapisuje unikalną wartość, co umożliwia identyfikowanie operacji zapisu poprzez tą wartość

Definicja historii

Historia lokalna (procesu p_i)
Zbiór liniowo uporządkowany $h_i = (O_i, \rightarrow_i)$

Historia globalna
Zbiór częściowo uporządkowany $h = (O, \rightarrow)$

Obraz historii h w procesie p_i
Zbiór liniowo uporządkowany $hv_i = (O_i \cup OW, \mapsto_i)$

Obraz historii h
Kolekcja obrazów procesów: $hv = \langle\, hv_1, hv_2, \ldots, hv_n \,\rangle$

Spójność sekwencyjna

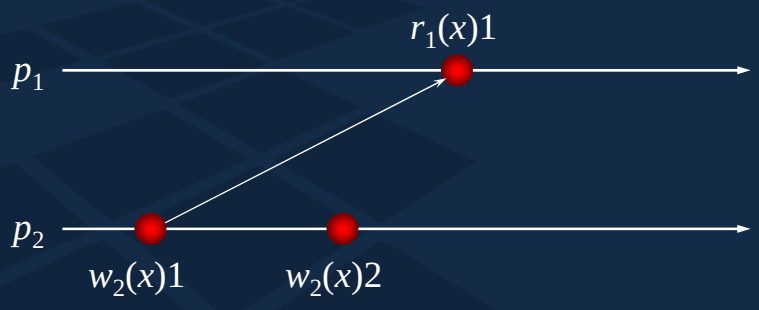
Obraz hv historii h musi spełniać następujące warunki:

$$\forall_{o1,\, o2 \,\in O_i \cup OW} \left(\left(\exists_{j=1..n} o1 \rightarrow_j o2 \right) \Rightarrow o1 \mapsto_i o2 \right)$$

$$\forall_{w1,\, w2 \,\in OW} \left(\forall_{i=1..n} w1 \mapsto_i w2 \vee \forall_{i=1..n} w2 \mapsto_i w1 \right)$$

Spójność sekwencyjna – przykład

hv_1: $w_2(x)1 \mapsto_1 r_1(x)1 \mapsto_1 w_2(x)2$



hv_2: $w_2(x)1 \mapsto_2 w_2(x)2$

Algorytm *fast-read*

- Protokół spójności dla modelu sekwencyjnego
- Algorytm dla procesu p_i

- **upon read(*x*)**
  return $M_i[x]$
- **upon write(*x*, *v*)**
  atomic_broadcast U(*x*, *v*)
  wait
- **upon receipt of U(*x*, *v*) from p_k**
  $M_i[x] := v$
  if $k = i$
  signal
  end if

Algorytm *fast-write*

- **upon read(*x*)**
  if $num_i \neq 0$
    wait
  end if
  return $M_i[x]$

- **upon write(*x*, *v*)**
  $num_i := num_i + 1$
  FIFO_atomic_broadcast U(*x*, *v*)

- **upon receipt of U(*x*, *v*) from p_k**
  $M_i[x] := v$
  if $k = i$
    $num_i := num_i - 1$
    if $num_i = 0$
      signal
    end if
  end if

Spójność atomowa

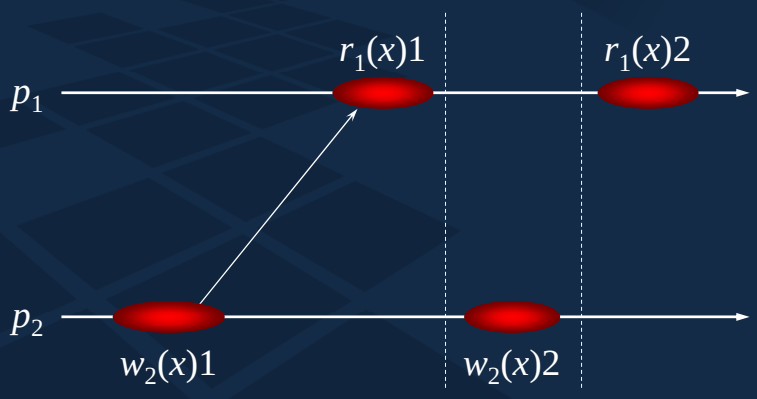
Obraz *hv* historii *h* musi spełniać następujące warunki:

$$\forall_{o1,\, o2 \,\in O_i \cup OW} \left(\left(\exists_{j=1..n} o1 \rightarrow_{RT} o2 \right) \Rightarrow o1 \mapsto_i o2 \right)$$

$$\forall_{w1,\, w2 \,\in OW} \left(\forall_{i=1..n} w1 \mapsto_i w2 \vee \forall_{i=1..n} w2 \mapsto_i w1 \right)$$

$o1 \rightarrow_{RT} o2$ *o1* kończy się **w czasie rzeczywistym**, zanim zaczyna się *o2*

Spójność atomowa — przykład



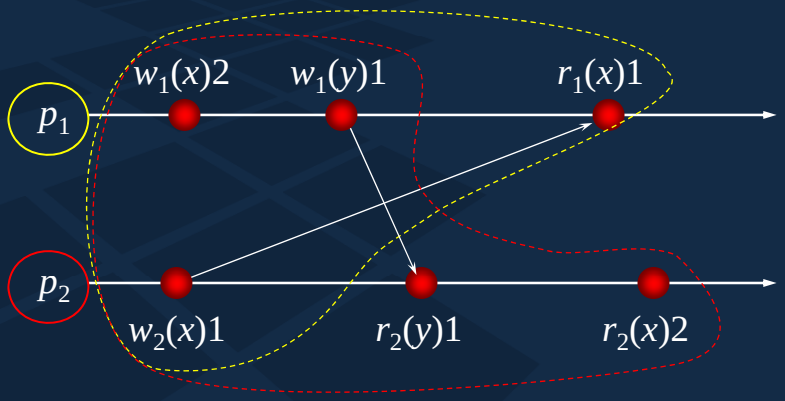
Spójność przyczynowa

Obraz *hv* historii *h* musi spełniać następujący warunek:

$$\forall_{o1,\, o2 \in O_i \cup OW} \left(o1 \rightarrow o2 \Rightarrow o1 \mapsto_i o2 \right)$$

Spójność przyczynowa — przykład

hv_1: $w_1(x)2 \mapsto_1 w_1(y)1 \mapsto_1 w_2(x)1 \mapsto_1 r_1(x)1$



hv_2: $w_2(x)1 \mapsto_2 w_1(x)2 \mapsto_2 w_1(y)1 \mapsto_2 r_2(y)1 \mapsto_2 r_2(x)2$

Protokół dla spójności przyczynowej

Algorytm dla procesu p_i

- **upon read(x)**
  return $M_i[x]$
- **write(x, v)**
  $M_i[x] := v$
  causal_broadcast U(x, v)
- **upon receipt of U(x, v) from p_k**
  if $k \neq i$
  $M_i[x] := v$
  end if

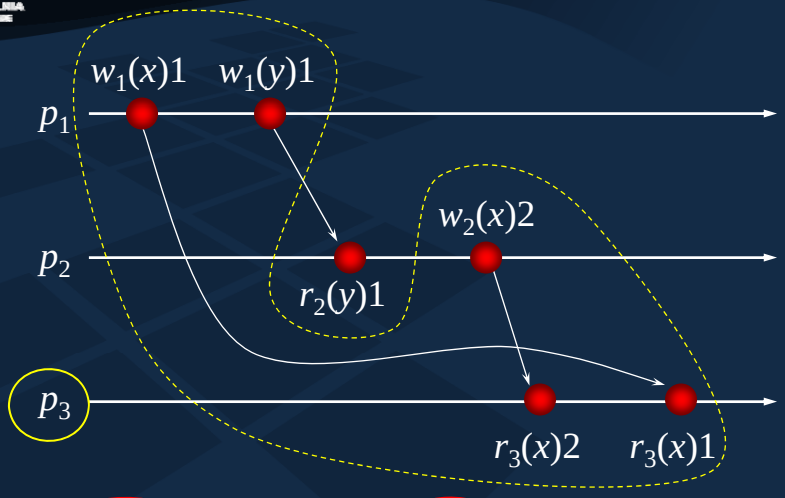
Spójność PRAM

PRAM — pipelined RAM

Obraz *hv* historii *h* musi spełniać następujący warunek:

$$\forall_{o1,\, o2 \in O_i \cup OW} \left(\left(\exists_{j=1..n} o1 \rightarrow_j o2 \right) \Rightarrow o1 \mapsto_i o2 \right)$$

Spójność PRAM — przykład



hv_3: $w_2(x)2 \mapsto_3 r_3(x)2 \mapsto_3 w_1(x)1 \mapsto_3 r_3(x)1 \mapsto_3 w_1(y)1$

Protokół dla spójności PRAM

Algorytm dla procesu p_i

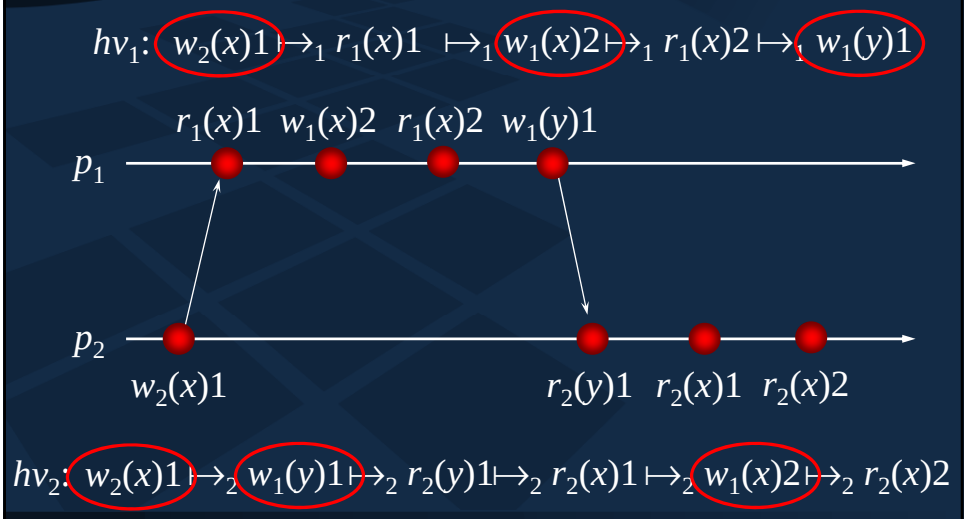
- **upon read(x)**
  return $M_i[x]$
- **upon write(x, v)**
  $M_i[x] := v$
  FIFO_broadcast U(x, v)
- **upon receipt of U(x, v) from p_k**
  if $k \neq i$
  $M_i[x] := v$
  end if

Spójność podręczna

Obraz *hv* historii *h* musi spełniać następujący warunek:

$$\forall_{x \in X} \quad \forall_{w1,\, w2 \in OW \cap O|x} \left(\forall_{i=1..n} w1 \mapsto_i w2 \vee \forall_{i=1..n} w2 \mapsto_i w1 \right)$$

Spójność podręczna — przykład



Protokół dla spójności podręcznej (1)

Algorytm *fast-read* dla procesu p_i:

- **upon read(*x*)**
  return $M_i[x]$
- **upon write(*x*, *v*)**
  atomicx_broadcast U(*x*, *v*)
  wait

- **upon receipt of U(*x*, *v*) from p_k**
  $M_i[x] := v$
  if $k = i$
  signal
  end if

Protokół dla spójności podręcznej (2)

Algorytm *fast-write* dla procesu p_i:

- **upon read(*x*)**
  if $num_i[x] \neq 0$
    wait
  end if
  return $M_i[x]$

- **upon write(*x*, *v*)**
  $num_i[x] := num_i[x] + 1$
  FIFOx_atomicx_broadcast U(*x*,*v*)

- **upon on receipt of U(*x*,*v*) from p_k**
  $M_i[x] := v$
  if $k = i$
    $num_i[x] := num_i[x] - 1$
    if $num_i[x] = 0$
      signal
    end if
  end if

Spójność procesorowa

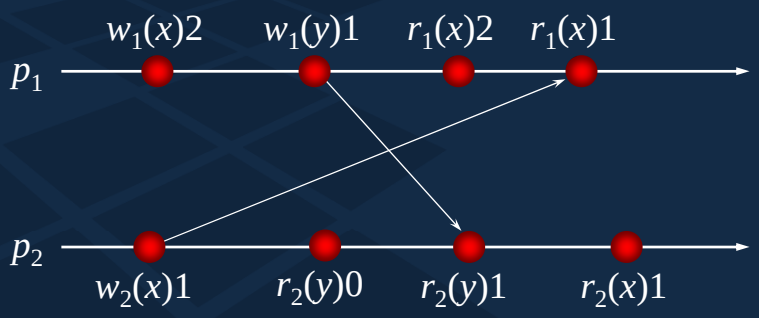
Obraz *hv* historii *h* musi spełniać następujące warunki (PRAM + spójność podręczna):

$$\forall_{x \in X} \; \forall_{w1,\, w2 \in OW \cap O|x} \left(\forall_{i=1..n} w1 \mapsto_i w2 \vee \forall_{i=1..n} w2 \mapsto_i w1 \right)$$

$$\forall_{o1,\, o2 \in O_i \cup OW} \left(\left(\exists_{j=1..n} o1 \rightarrow_j o2 \right) \Rightarrow o1 \mapsto_i o2 \right)$$

Spójność procesorowa — przykład

hv_1: $w_1(x)2 \mapsto_1 w_1(y)1 \mapsto_1 r_1(x)2 \mapsto_1 w_2(x)1 \mapsto_1 r_1(x)1$



hv_2: $w_1(x)2 \mapsto_2 w_2(x)1 \mapsto_2 r_2(y)0 \mapsto_2 w_1(y)1 \mapsto_2 r_2(y)1 \mapsto_2 r_2(x)1$

Protokół dla spójności procesorowej

Algorytm *fast-write* dla procesu p_i:

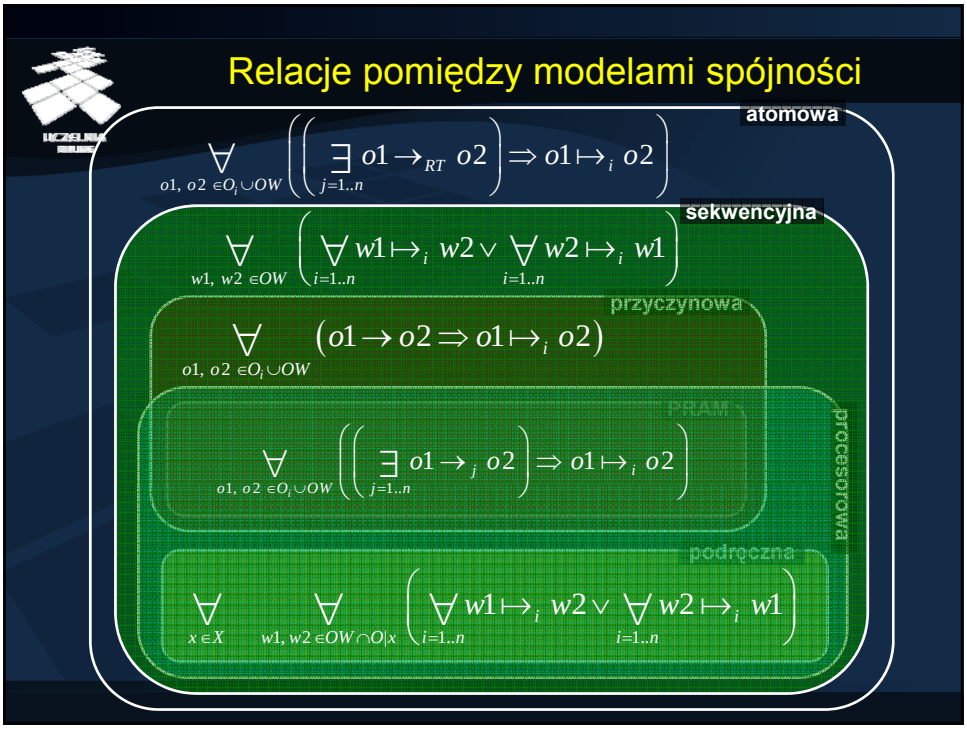
- **upon read(*x*)**
  if $num_i[x] \neq 0$
    wait
  end if
  return $M_i[x]$

- **upon write(*x*, *v*)**
  $num_i[x] := num_i[x] + 1$
  FIFO_atomicx_broadcast U(*x*, *v*)

- **upon receipt of U(*x*, *v*) from p_k**
  $M_i[x] := v$
  if $k = i$
    $num_i[x] := num_i[x] - 1$
    if $num_i[x] = 0$
      signal
    end if
  end if

Naruszenie porządku przyczynowego

Relacje pomiędzy modelami spójności



Zadanie

W jakich modelach spójności wskazana operacja odczytu $r_3(x)v$

1. zwróci wartość $v = 1$,
2. zwróci wartość $v = 2$,
3. zwróci wartość $v = 3$,