Rozproszona pamięć dzielona - 2

Rozproszona pamięć dzielona ze stronami

NORMA (**NO R**emote **M**emory **A**ccess) - wielokomputery, czyli np. stacje robocze połączone w sieć lokalną: żaden procesor nie ma bezpośrednio dostępu do pamięci innego procesora
Różnice między maszynami NUMA i NORMA
Można zrealizować rozpr. pamięć dzieloną w wielokomputerze emulując pamięć podręczną wieloproc. przy użyciu MMU i SO
Jaką wybrać jednostkę transmisji: słowo, blok (kilka słów), stronę, segment (kilka stron)?

- transmisja większych jednostek jest b. efektywna czasowo
- większe transmisje oznaczają jednak większe obciążenie sieci
- większa jednostka jest b. narażona na *fałszywe współdzielenie* (strona zawiera dwie niezwiązane ze sobą zmienne, używane przez dwa różne procesory, i kursuje intensywnie między nimi)

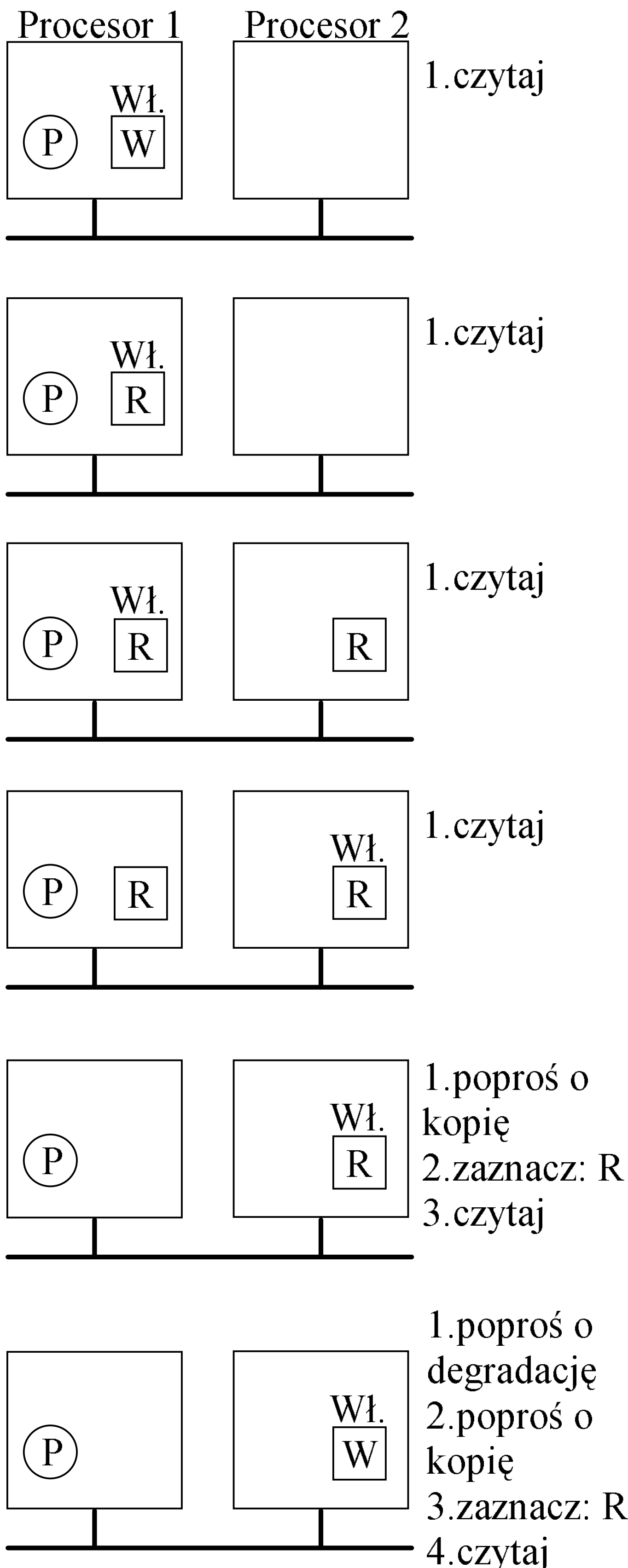
Jak zapewnić zgodność wielu kopii?

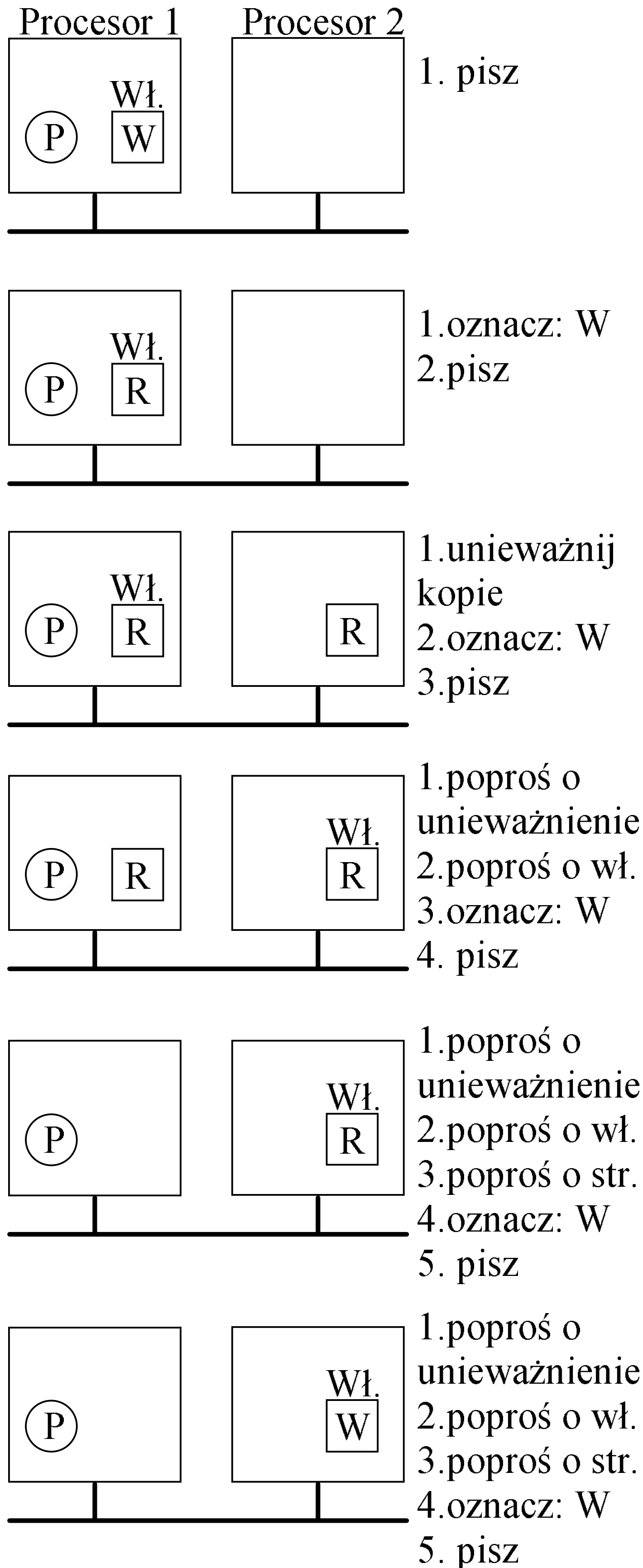
- w wieloprocesorze dopuszczalne rozwiązanie w przypadku pisania do strony istniejącej w wielu kopiach to:
  - uaktualnienie wszystkich kopii
  - unieważnienie wszystkich kopii z wyjątkiem jednej
- w wielokomputerze
  - uaktualnienie jest zbyt trudne do zrealizowania
  - zwykle stosuje się unieważnienie

Przykładowy protokół (***sequential consistency***)

- W każdej chwili każda strona jest w stanie R (do czytania) lub W (do czytania i pisania) - stan może się zmieniać
- Każda strona ma właściciela (proces, który ostatnio na niej pisał)
- Jeśli strona jest w stanie W, to istnieje tylko jedna kopia odwzorowana do przestrzeni adresowej właściciela

- Jeśli strona jest w stanie R, to właściciel ma kopię, ale również inne procesy mogą ją mieć





1. Jak znaleźć właściciela?

- przez rozgłaszanie
- rozgłaszanie, ale z pytaniem o właściciela idą dodatkowe informacje (czytanie czy pisanie, czy jest potrzebna kopia); właściciel przesyła komunikat przekazujący prawo własności i, w razie potrzeby, stronę
- rozgłaszanie jest kosztowne (niepotrzebnie absorbuje uwagę wielu procesorów). Można wyznaczyć jednego (lub wielu) zarządcę stron, który pamięta kto jest właścicielem każdej strony. Ew. każdy procesor może pamiętać kto jest *potencjalnym* właścicielem

2. Jak znaleźć wszystkie kopie?

- przez rozgłaszanie komunikatu z numerem strony. Akceptowalne, gdy rozgłaszanie jest niezawodne (komunikaty nie giną)
- właściciel lub zarządca stron przechowuje listę wszystkich kopii. Wysyła się do każdego procesu posiadającego kopię żądanie unieważnienia i czeka na potwierdzenie

3.Wymiana stron

- na sprowadzaną stronę może zabraknąć miejsca
- którą stronę usunąć? Np. LRU. Dobrym kandydatem jest strona, której właścicielem jest inny proces lub strona posiadająca kopie, której właścicielem jest dany proces - trzeba jednak przekazać innemu procesowi prawo własności. Wpp strona, która nie ma kopii - można ją wysłać na dysk lub przekazać innemu procesorowi (np. przypisać każdej stronie procesor domowy)

4.Synchronizacja

- w multiprocesorze do implementacji wzajemnego wykluczania często używa się instrukcji Test-And-Set i prostej zm. globalnej
- w systemie DSM takie rozwiązanie może być bardzo nieefektywne, gdy kilka procesów próbuje równocześnie wejść

do sekcji krytycznej - każdy ściąga całą stronę, by zmienić wartość jednej zmiennej. Możliwe rozwiązanie: zarządca synchronizacji (mniejszy ruch w sieci, ale centralizacja)

Rozproszona pamięć dzielona z dzielonymi zmiennymi

Procesy mogą współdzielić zmienne i struktury danych
Problem: jak utrzymywać rozproszoną bazę danych współdzielonych zmiennych, potencjalnie z wieloma kopiami?

System **Munin**

- Kompilator umieszcza każdą dzieloną zmienną (lub grupę zmiennych) na osobnej stronie
- Różne procesy tego samego programu mogą się wykonywać na różnych procesorach, ale bez możliwości migrowania
- Odwołanie do nieobecnej zmiennej dzielonej powoduje błąd braku strony i przekazanie sterowania do SO
- Trzy klasy zmiennych:

1. *zwykłe* - tylko proces, który utworzył taką zmienną może ją czytać i zapisywać
2. *dzielone* - są widoczne dla wielu procesów, ale należy z nich korzystać jedynie w rejonach krytycznych; deklaruje się je jako dzielone, ale czyta i zapisuje w zwykły sposób
3. *synchronizacyjne* - dostęp do nich jest możliwy tylko porzez specjalne procedury systemowe (np. lock(L), unlock(L)):

proces 1:

```
lock(L);                  - wejście do rejonu krytycznego
a = 1; b = 2; c = 3;      - zmiana wartości zmiennych dzielonych
unlock(L);                - wyjście z rejonu krytycznego; propagacja
                            zmian wartości zm. a, b, c do maszyn
                            zawierających kopie (release consistency)
```

- Próba dostępu na innych maszynach do zmiennych dzielonych poza rejonem krytycznym (w czasie gdy jest w nim proces 1) daje niezdefinowane wyniki
- Programista może zakwalifikować zmienną do jednej z nast. kategorii (każda maszyna przechowuje *katalog* zmienych z informacją o: kategorii, istnieniu lokalnej kopii i ew. właścicielu):

1. *tylko do czytania*: system szuka w katalogu kto jest właścicielem i prosi go o kopię, sprzętowa ochrona przed zapisem
2. *migrująca*: używane wewnątrz rejonu krytycznego i chronione przez zmienne synchronizujące, migrują z maszyny do maszyny gdy następuje wejście i wyjście z rejonu krytycznego, nie posiadają kopii
3. *dzielona do pisania* : zmienne, które można bezpiecznie zapisywać w dwóch procesach w tym samym czasie (np. różne podtablice tej samej tablicy); inicjalnie oznaczone *tylko do czytania*; przy próbie zapisu SO tworzy kopię (*bliźniaka*), oznacza stronę jako brudną i pozwala na zapis; po zakończeniu SO porównuje stronę i bliźniaka słowo po słowie i wysyła listę różnic do wszystkich zainteres. procesów; następnie przywraca tryb *tylko do czytania*; odbiorca listy sprawdza swoją kopię: jeśli sam nie robił modyf., to akceptuje zmiany, wpp sprawdza słowo po słowie: jeśli lokalne słowo było modyf., a przysłane nie, to akceptuje przysłane; jeśli oba słowa były modyf., to zgłasza błąd
4. *konwencjonalna*: każda zapisywana strona jest dostępna tylko w jednej kopii i przesyłana od procesu do procesu na żądanie

System **Midway**

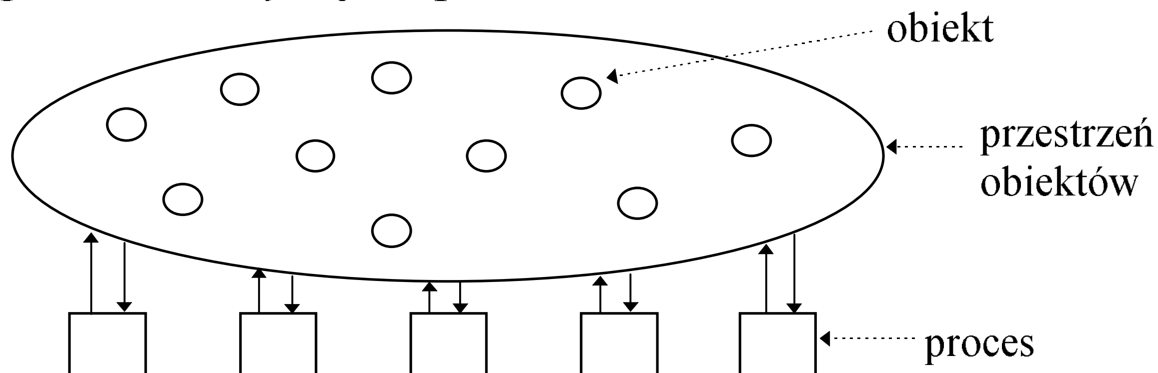
- Wątki mają tą samą liniową przestrzeń adresową, która zawiera dane prywatne i dzielone; mogą się wykonywać równolegle na różnych maszynach
- Dostęp do zmiennych dzielonych jest możliwy tylko wewnątrz specjalnej sekcji krytycznej, chronionej przez specjalną zmienną

synchronizacyjną; zmienną dzieloną trzeba jawnie związać ze zmienną synchronizacyjną poprzez wywołanie procedury

- Żeby wejść do sekcji krytycznej, proces wywołuje lock z żądaniem **wyłącznego dostępu** (*exclusive lock*) lub **niewyłącznego dostępu** (*nonexclusive lock*). SO zakłada blokadę i równocześnie uaktualnia wartości wszystkich zmiennych dzielonych związanych z daną zmienną synchr. (***entry consistency***); podczas zdejmowania blokady SO nie uaktualnia wartości zmiennych
- Zakładanie blokady wymaga kontaktu z właścicielem klucza, czyli procesem, który jako ostatni uzyskał wyłączny dostęp; następuje przekazanie prawa do wyłącznego dostępu (lub proces musi czekać). Równocześnie poprzedni właściciel przesyła aktualne wartości zmiennych dzielonych
- Jak system rozpoznaje co i kiedy uległo zmianie: z każdą zmienną dzieloną pamięta się czas ostatniej zmiany (system zegarów logicznych!)
- Protokół zapewnia dobrą wydajność (mało komunikatów!), ale jest mniej wygodny dla programisty (złożony i błędogenny interfejs)

Rozproszona pamięć dzielona z obiektami

Procesy na wielu maszynach współdzielą abstrakcyjną przestrzeń wypełnioną obiektami. Zarządza nimi automatycznie system wspomagający wykonanie programu (ang. *runtime system*). W szczególności decyduje o położeniu obiektów



Korzyści:

1. Bardziej modularne podejście niż w innych technikach
2. Bardziej elastyczna implementacja, gdyż dostęp do obiektów dzielonych jest kontrolowany
3. Można łatwo zintegrować synchronizację i dostęp

Linda

Możliwość rozbudowania dowolnego języka o zestaw prymitywów synchronizacyjnych (np. C-Linda, Fortran-Linda)

Przestrzeń krotek i operacje na krotkach:

("abc", 2, 5)
("macierz-1", 1, 6, 3.14)
("rodzina", "jest-siostrą", "Alicja", "Elżbieta")

out("abc", 2, 5) - wstawia krotkę do przestrzeni
in("abc", 2, i) - wyjmuje krotkę z przestrzeni
read("abc", 2, i) - sprawdza istnienie krotki w przestrzeni

Paradygmat programowania: *model powielonych pracowników*

out("pula-zleceń", "zlecenie")
in("pula-zleceń", moje-zlecenie)

- Implementacja:

Preprocesor czyta program w Lindzie i przekształca go w program w języku bazowym. Operacje na krotkach są realizowane podczas wykonania programu przez system wspomagający Lindy

- Problemy:

1. Jak symulować adresowanie asocjacyjne bez masowego przeszukiwania?
2. Jak rozpraszać krotki pomiędzy maszynami i jak je później odnajdywać?

- Każda krotka ma *sygnaturę typu* (uporządkowana lista typów pól). Pierwsze pole jest zwykle napisem. Dzieli się przestrzeń krotek na podprzestrzenie o tej samej sygnaturze i pierwszym polu. Pozwala to ograniczyć (a więc usprawnić) przeszukiwanie.

Każdą podprzestrzeń organizuje się jako tablicę mieszającą z *i*-tym polem krotki jako kluczem

- W multiprocesorze podprzestrzenie można zaimplementować jako tablice mieszające w pamięci globalnej. Na czas wykonania in i out blokuje się dostęp do podprzestrzeni
- W multikomputerze: Jeśli jest dostępne rozgłaszanie, to powiela się w całości podprzestrzenie na wszystkich maszynach. Podczas realizacji out rozgłasza się nową krotkę i wprowadza do każdej podprzestrzeni. Podczas realizacji in pobiera się ją z lokalnej podprzestrzeni i usuwa z pozostałych maszyn (dwufazowe wykonanie, ang. *two-phase commit*)
- Inne rozwiązanie: out wykonuje się lokalnie, in powoduje rozgłoszenie wzorca krotki. Jeśli zostanie przysłana więcej niż jedna krotka, to nadmiarowe traktuje się jak lokalne out
- Inne rozwiązanie: częściowe powielanie (wszystkie maszyny tworzą logiczną macierz: out jest rozgłaszane wzdłuż wiersza tej macierzy, a in wzdłuż kolumny)

Orca

- Współdzielone *obiekty*
- *Dozory*: jeśli wszystkie dozory danej operacji są fałszywe, to wstrzymuje się proces do czasu, aż jeden stanie się prawdziwy. Następnie wykonuje się blok instrukcji związany z prawdziwym dozorem

```
Object implementation stack;
  top: integer;
  stack: array [integer 0..N-1] of integer;

  operation push (item:integer);
  begin
    stack[top] := item;
    top := top+1;
  end;
```

```
 operation pop(): integer;
  begin
    guard top > 0 do
      top := top-1;
      return stack[top];
    od
  end;
begin
  top := 0;
end;
```

- Tworzenie procesów: fork tworzy na wskazanym procesorze proces wykonujący wskazaną procedurę; można mu przekazać obiekt jako parametr. System wspomagający wykonanie programu realizuje iluzję pamięci dzielonej. Operacje na obiektach dzielonych są atomowe, wzajemnie wykluczające się i sekwencyjnie zgodne (jeśli dwa niezależne procesy wstawią na stos, odpowiednio, 3 i 4, to każdy inny proces wykonujący top, zauważy na stosie to samo).

```
s: stack;
for i in 1 .. n do fork proc(s) on i; od;
```

- Każdy z obiektów może być (niezależnie od innych) w stanie: *pojedynczy* lub *powielony*; stan obiektu może się zmieniać. Obiekt powielony jest obecny na wszystkich maszynach, na których znajduje się używający go proces. Możliwe przypadki:

1. Operacja na lokalnym obiekcie pojedynczym
   zablokowanie obiektu; wykonanie operacji; odblokowanie obiektu
2. Operacja na zdalnym obiekcie pojedynczym
   wykonanie RPC do zdalnej maszyny z poleceniem wykonania operacji (czytania bądź pisania)
3. Operacja na powielonym obiekcie - czytanie
   Istnieje lokalna kopia, więc wykonuje się lokalne czytanie
4. Operacja na powielonym obiekcie - pisanie

Jeśli jest dostępne niezawodne rozgłaszanie z całkowitym porządkiem, to system wspomagający rozgłasza nazwę obiektu, operację i parametry i czeka zablokowany do zakończenia wszystkich operacji. Jeśli nie ma rozgłaszania, uaktualnia się kopie za pomocą *dwufazowego algorytmu z kopią główną*: proces wysyła komunikat do głównej kopii obiektu, blokuje ją i uaktualnia. Następnie ta kopia wysyła komunikaty do pozostałych kopii z żądaniem założenia blokady. Gdy dostaje potwierdzenie od wszystkich, to oryginalny proces rozpoczyna drugą fazę, wysyła do wszystkich komunikat z żądaniem uaktualnienia i zdjęcia blokady

Porównanie

- IVY próbuje symulować wieloprocesor wykonując stronicowanie poprzez sieć. Zapewnia sekwencyjną zgodność i może wykonywać bez modyfikacji programy przygotowane na wieloprocesory. Problemem jest natomiast wydajność
- Munin i Midway próbują poprawić wydajność wymagając od programisty wskazania dzielonych danych i zapewniając słabsze modele zgodności. Munin wspiera *zgodność zwalniania*, a Midway *wejścia*. Munin wspiera cztery typy dzielonych zmiennych, a Midway jedną. Munin używa sprzętu MMU do wykrywania zapisu, a Midway MMU lub kompilatora
- W systemie Midway i Munin programiści muszą wykonać więcej pracy w celu zapewnienia synchronizacji i zgodności; w Lindzie i Orce synchronizację zapewnia system wspomagający czasu wykonania