Pamięć wirtualna

Ostatnia modyfikacja: 04.05.2021

Adaptacja slajdów Silberschatz, Galvin, Gagne do podręcznika
„Podstawy systemów operacyjnych”

Spis treści

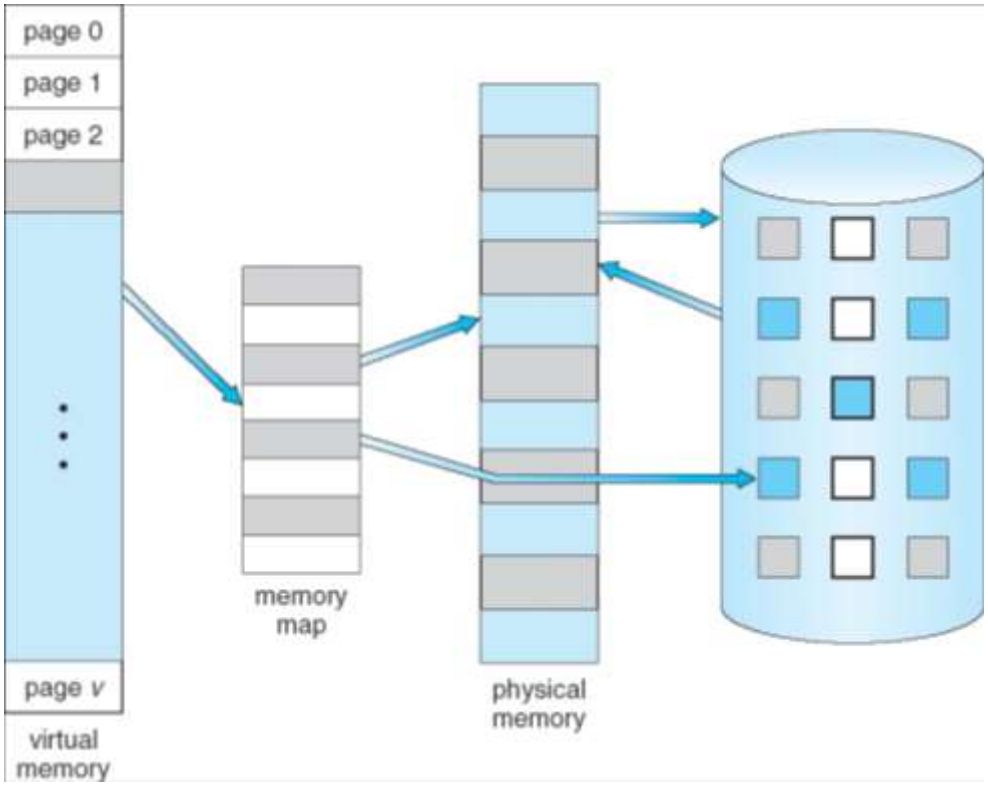
Podstawy

- Pamięć wirtualna – technika umożliwiająca odseparowanie pamięci logicznej użytkownika od pamięci fizycznej.
  - Tylko część programu musi być w pamięci operacyjnej w czasie jego wykonywania.
  - Przestrzeń adresów logicznych może być dużo większa niż przestrzeń adresów fizycznych.
  - Rozmiar pamięci fizycznej nie eliminuje możliwości wykonywania dużych programów
  - W tej samej pamięci fizycznej zmieści się więcej (fragmentów) programów
  - Potrzeba mniej operacji wejścia/wyjścia by załadować, czy wytoczyć proces

- Pamięć wirtualna może być implementowana jako:
  - stronicowanie na żądanie
  - segmentacja na żądanie

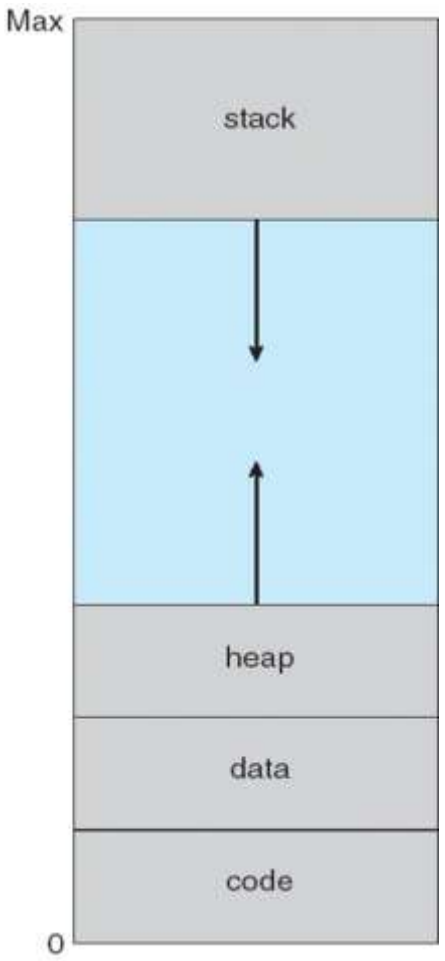
Pamięć wirtualna, a pamięć fizyczna

- Memory management unit (MMU) odwzorowuje adresy logiczne w fizyczne. Ideę tę, dla przypadku stronicowania, pokazano poniżej. Podobnie realizowane jest odwzorowanie w przypadku segmentacji (zamiast adresu postaci: adres początku strony +przesunięcie na stronie jest adres=adres początku segmentu+przesunięcie w segmencie).

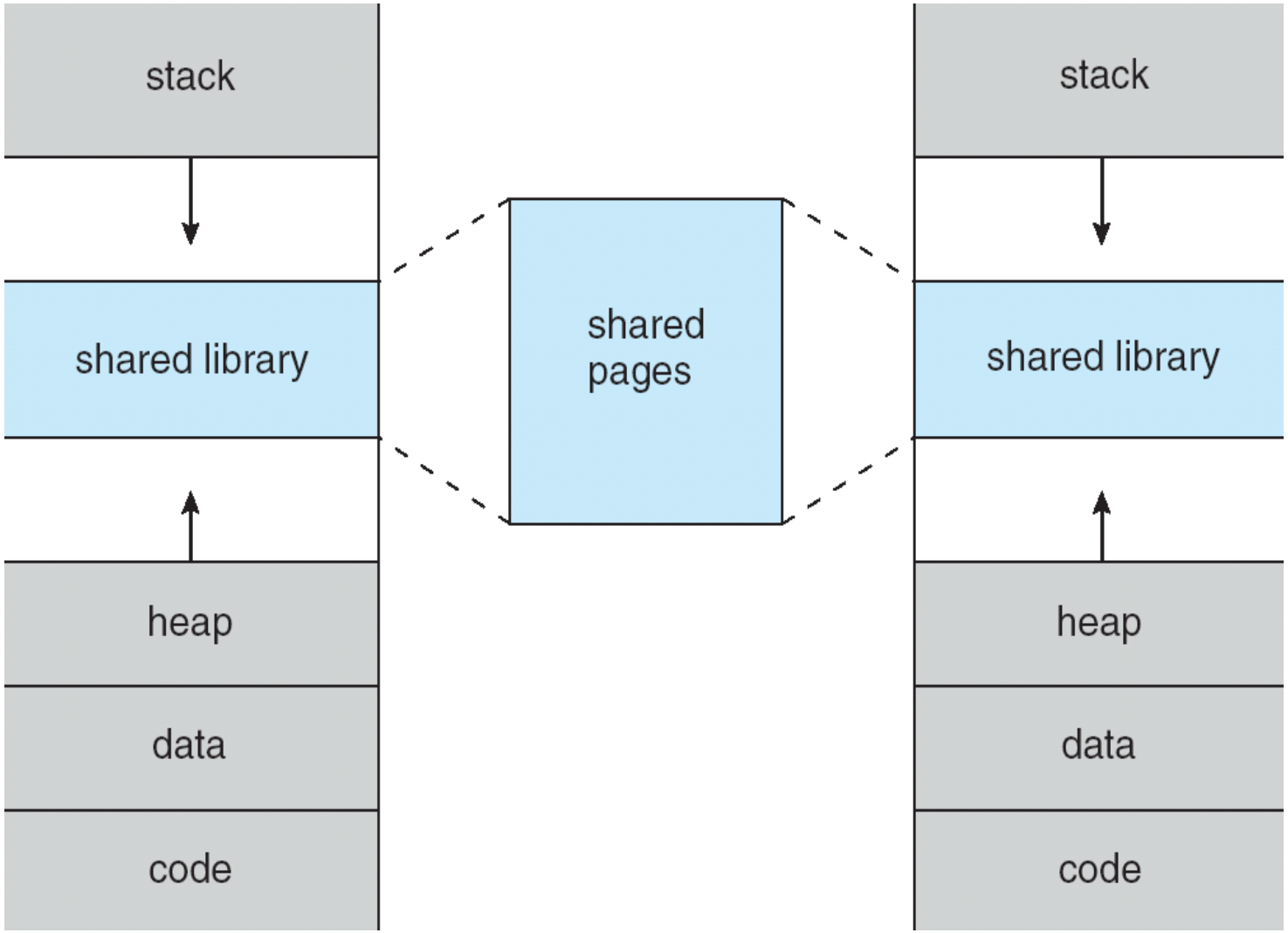


Przykładowy układ przestrzeni wirtualnej procesu

- Stos i sterta zmieniają rozmiar w czasie życia procesu – pokazany układ umożliwia wykorzystanie całej wolnej pamięci.
- W wolnej przestrzeni umieszczone są też pliki odwzorowywane w pamięci oraz dynamicznie ładowane biblioteki. Utrudnia to zarządzanie pamięcią.
- Segmenty/strony procesu mogą być współdzielone w trybach RWX (kod bibliotek, segmenty współdzielonych danych). Współcześnie funkcje `fork()`, wykorzystują możliwość współdzielenia wirtualnej przestrzeni adresowej rodzica przez potomka – w trybie *copy-on-write (COW).*

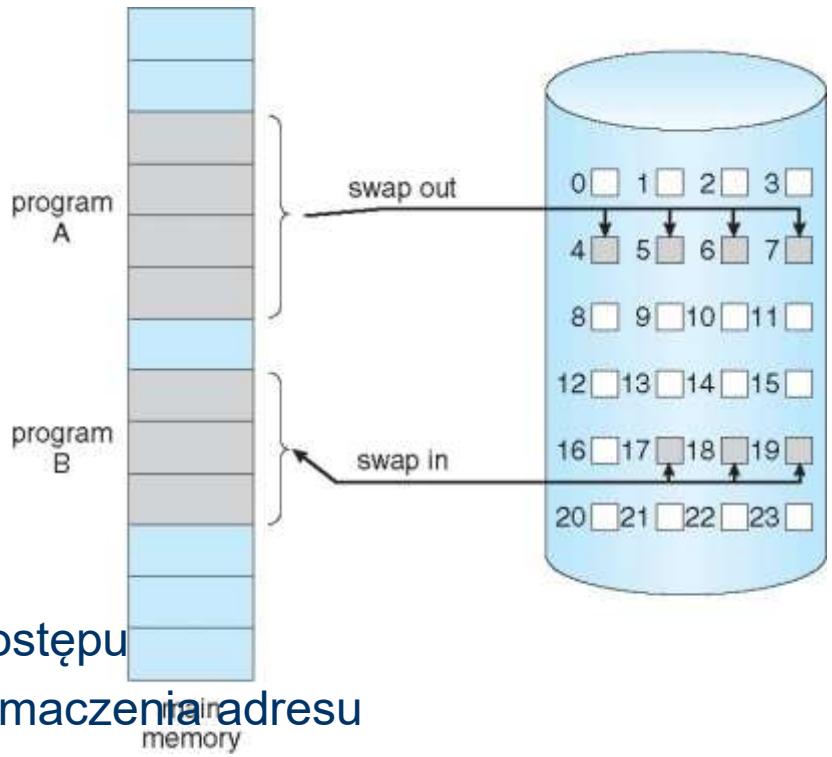


Wykorzystanie pamięci wirtualnej do współdzielenia dostępu



Stronicowanie na żądanie

- Strona jest kopiowana do pamięci tylko wtedy, gdy jest potrzebna.
  - mniej operacji wejścia/wyjścia (nieużywane strony na dysku)
  - mniejsze zużycie pamięci operacyjnej
  - szybszy system (jeśli mniej stron do kopiowania z dysku)
  - więcej procesów, użytkowników
- Wymiana stron (a nie całego procesu)



- Strona jest potrzebna procesorowi ⇒ próba dostępu
  - strona w pamięci ⇒ użyty mechanizm tłumaczenia adresu
  - w przeciwnym razie błąd (pułapka) braku strony (*page-fault*):
    - jeżeli odwołanie nielegalne ⇒ *abort*, w przeciwnym razie kopiowanie potrzebnej strony do pamięci
- **Leniwa wymiana (*lazy swapping*)** – strona nie jest nigdy kopiowana do pamięci - jeśli nie jest potrzebna
- **Pager** – systemowy proces (demon) wymiany stron

Problemy

- Kiedy zestaw stron procesu ma być ładowany do pamięci proces stronicujący (**pager**) zgaduje które procesy będą potrzebne i kopiuje tylko te wybrane
- Dla realizacji potrzeb pamięci potrzebne są dodatkowe własności MMU
- Jeżeli strony potrzebne w czasie wykonania procesu są już w pamięci fizycznej – sytuacja jest taka sama jak dla zwykłego stronicowania pamięci.
- Jeżeli strona potrzebna procesowi nie znajduje się w pamięci fizycznej
  - Trzeba wykryć brak i załadować stronę do pamięci
    - nie zmieniając (istotnie) zachowania programu
    - nie wymagając specjalnego przygotowania kodu przez programistę aplikacji

Bit poprawności

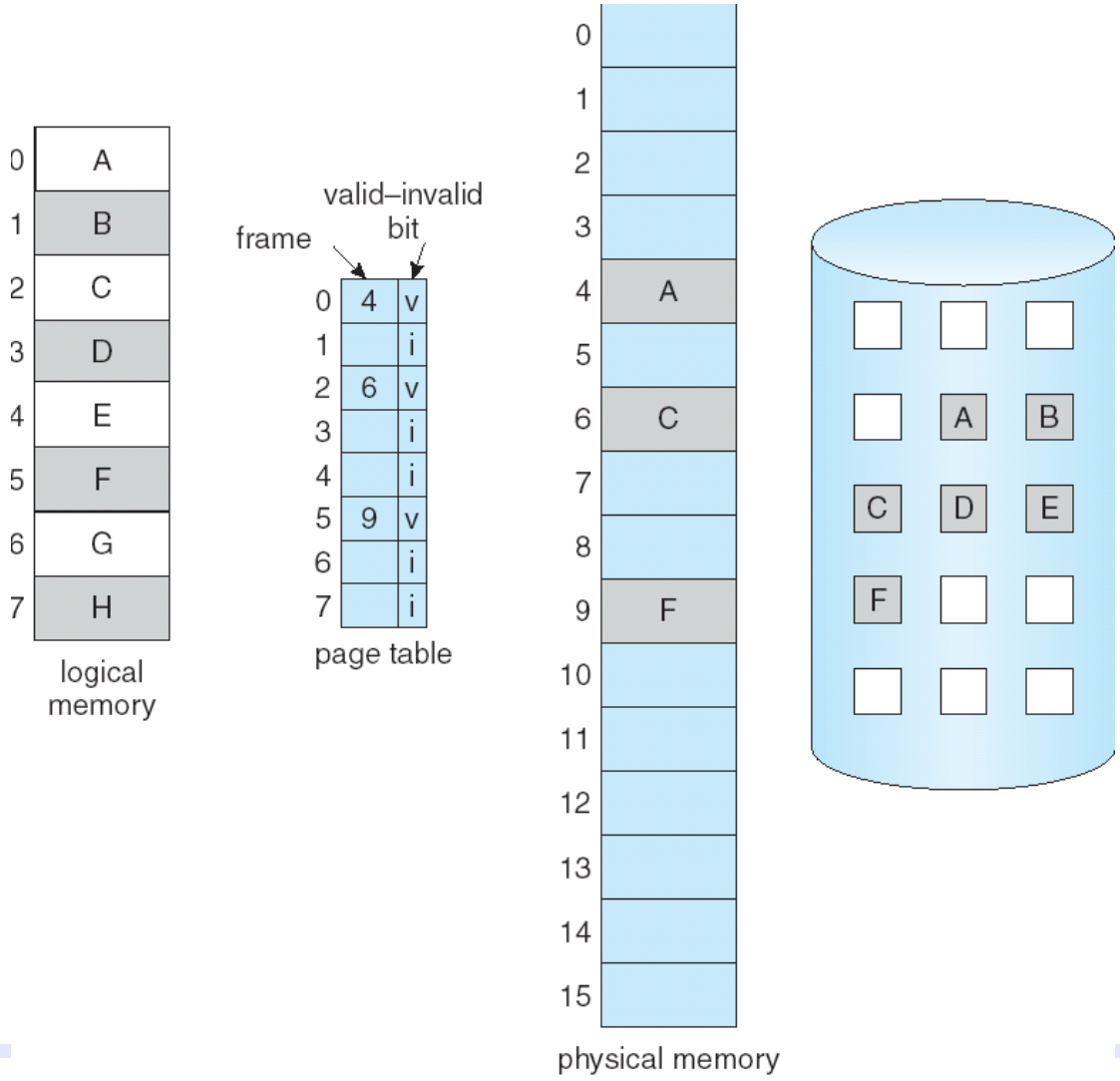
- Każda pozycja tablicy stron może mieć skojarzony bit poprawności (*valid–invalid bit*) :1 ⇒ strona w pamięci operacyjnej, 0 ⇒ poza pamięcią
- Początkowo bit poprawności =0 dla całej tablicy stron.
- Przykład zawartości tablicy stron.

Nr ramki	Bit poprawności
	1
	1
	1
	1
	0
⋮	
	0
	0

Tablica stron

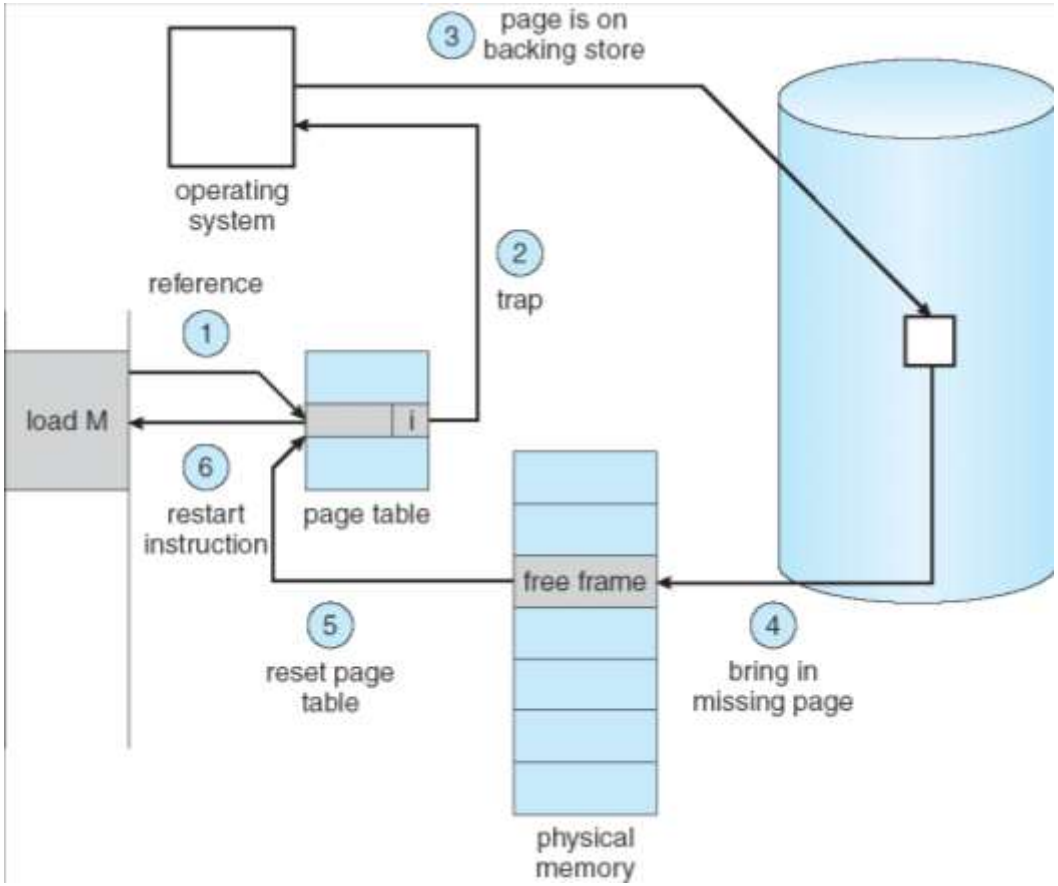
- Jeśli w czasie tłumaczenia adresu (log. na fiz.) bit poprawności =0, to sprzęt generuje **błąd braku strony**.

Przykładowa zawartość tablicy stron dla częściowego odwzorowania przestrzeni logicznej w pamięć operacyjną



Błąd braku strony

- Można rozpocząć wykonanie procesu bez żadnej strony w pamięci. Pierwsze odniesienie do strony wyzwoli pułapkę ⇒ błąd braku strony (*page-fault*).
- System operacyjny sprawdza:
  - jeśli nieprawidłowy adres (niedostępny dla procesu) ⇒ zakończenie (*abort)*
  - brak strony w pamięci. System
    - pozyskuje wolną ramkę,
    - kopiuje stronę do ramki,
    - zmienia zawartość tablicy stron, wstawiając aktualny numer ramki i ustawiając bit poprawności v=1
    - następnie wznawia wykonywanie ostatniego rozkazu, przerwanego wskutek tego, że zawierał odwołanie do niedozwolonego adresu .

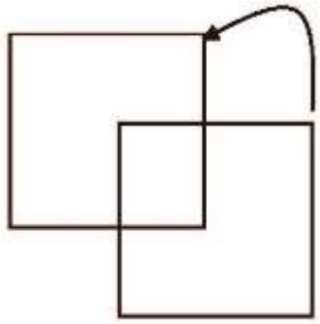


Uwagi

- Czyste stronicowanie na żądanie (***pure demand paging***)
  - Proces jest uruchamiany bez żadnej strony w pamięci
  - Pierwsze wskazanie adresu na każdej stronie, która nie została jeszcze załadowana do pamięci wywołuje błąd braku strony (*page fault*)
- Pojedyncza instrukcja procesora może wymagać dostępu do kilku stron, a więc generować kilka błędów braku strony
  - Przykład: instrukcja, która sumuje wartości dwóch liczb umieszczonych w pamięci, zapamiętując wynik w pamięci
  - **Lokalność odniesień do pamięci** zmniejsza liczbę wystąpień błędu braku strony.
- Wymagania sprzętowe stronicowania na żądanie:
  - Tablica stron z bitem poprawności dla każdej strony
  - Pamięć stronicowania (*swap device*)
  - Wznawianie rozkazu, który wywołał błąd braku strony
- Wejście/wyjście wymiany (*swap space I/O*) jest szybsze od obsługi plików – nawet przy użyciu tego samego urządzenia – obszar danych wymiany posługuje się dużymi blokami, a narzut na zarządzanie jest mały.

Restart instrukcji procesora

- Przykład instrukcji, która wykorzystuje kilka odniesień do pamięci: przeniesienie bloku pamięci:



  - Automatyczna inkrementacja/dekrementacja adresów
  - Czy restart powinien dotyczyć całej instrukcji? A jeśli obszar docelowy i źródłowy nie są rozłączne?

Scenariusz najgorszego przypadku dla stronicowania na żądanie

1. Pułapka braku strony przenosi sterowanie do trybu uprzywilejowanego
2. Zachowane są rejestry i pozostałe składniki stanu procesu
3. Wywoływana jest procedura obsługi pułapki błędu braku strony
4. Procedura obsługi sprawdza czy odniesienie do strony było legalne, a jeśli tak – określona jest lokalizacja strony na dysku.
5. Procedura obsługi zamawia odczyt stron z dysku do wolnej rami w pamięci.:
    1. Oczekiwanie na przyjęcie zlecenia przez sterownik dysku
    2. Oczekiwanie na znalezienie bloku na dysku (*device seek and/or latency time*)
    3. Inicjacja przesłania strony do ramki w pamięci (DMA)
6. W czasie oczekiwania na stronę procesor jest przydzielony innemu procesowi
7. Odebrano przerwanie wejścia/wyjścia po zakończeniu przesyłania strony
8. Zapamiętanie rejestrów i pozostałych składników stanu aktualnego proces
9. Wywołanie procedury obsługi przerwania wejścia/wyjścia
10. Modyfikacja tablicy stron i ew. innych danych tak, by odzwierciedlić obecność nowej strony w pamięci
11. Proces oczekuje na ponowny przydział procesora
12. Odtworzone są rejestry i pozostałe składniki stanu procesu (w tym tablica stron procesu), po czym wznowiona jest instrukcja procesora, która spowodowała błąd braku strony.

Średnia sprawność stronicowania na żądanie

- Prawdopodobieństwo błędu braku strony $0 \leq p \leq 1.0$
  - jeżeli $p = 0$, to brak błędów braku strony
  - jeżeli $p = 1$, to każde odniesienie do pamięci prowadzi do błędu braku strony
- Efektywny czas dostępu (EAT)

EAT = (1 – *p*) x czas dostępu do pamięci

+ *p* (czas obsługi *page fault*

+ [czas obsługi usuwania strony]

+ czas obsługi sprowadzania strony

+ czas restartowania przerwanej instrukcji

+ czas dostępu do pamięci)

Przykład stronicowania na żądanie

- Zał.: czas dostępu do pamięci = 200ns
- Średni czas obsługi błędu braku strony = 8 ms
- EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p) x 200 + p x (8,000,000+200)

  = 200 + p x 8,000,000 ns
- Jeśli jedno odniesienie na 1,000 wywołuje błąd braku strony, to

  EAT = 8.2 μ s.

  To oznacza 40-krotne spowolnienie dostępu!!
- Dla pogorszenia średniego czasu dostępu do pamięci < 10 %
  - 220 > 200 + 8,000,000 x p
    20 > 8,000,000 x p
  - $p < 2.5*10^{-6}$
  - < jeden błąd braku strony na 400,000 odniesień
- Jaka jest wówczas maksymalna średnia częstość błędów braku strony?

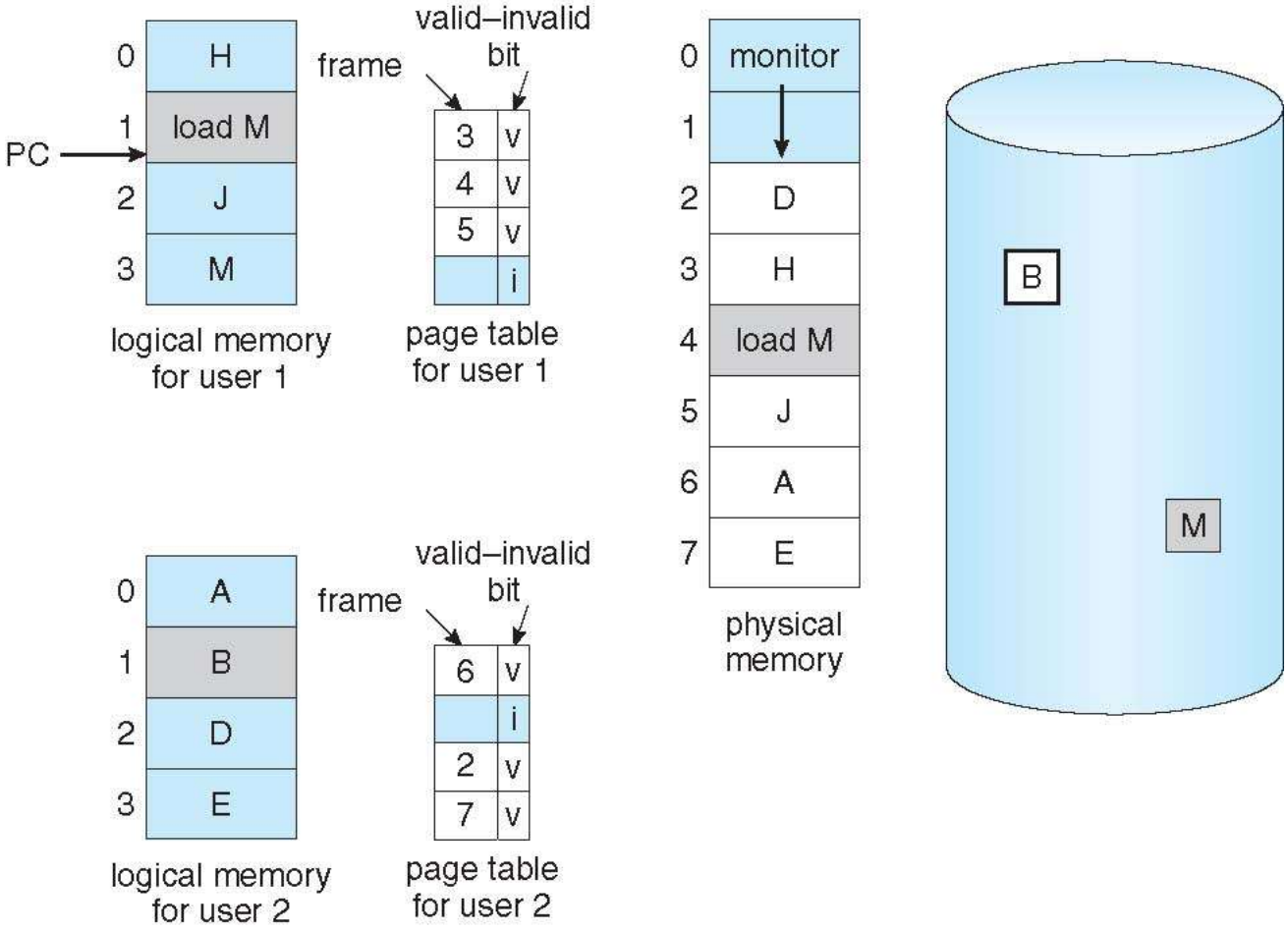
Zastępowanie stron

- **Zastępowanie stron** zapobiega nadmiernemu zapełnieniu („nadprzydziałowi”, *over-allocation*) pamięci przez proces.

- **Bit modyfikacji strony** (*dirty bit)* zmniejsza liczbę przesłań stron zmodyfikowanych w pamięci operacyjnej na urządzenie wymiany.

- **Zastępowanie stron** jest podstawą stronicowania na żądanie. Dopełnia ono rozdzielenia pamięci logicznej od fizycznej. Umożliwia programowi dostęp do dużej pamięci wirtualnej przy wykorzystaniu niedużej pamięci fizycznej.

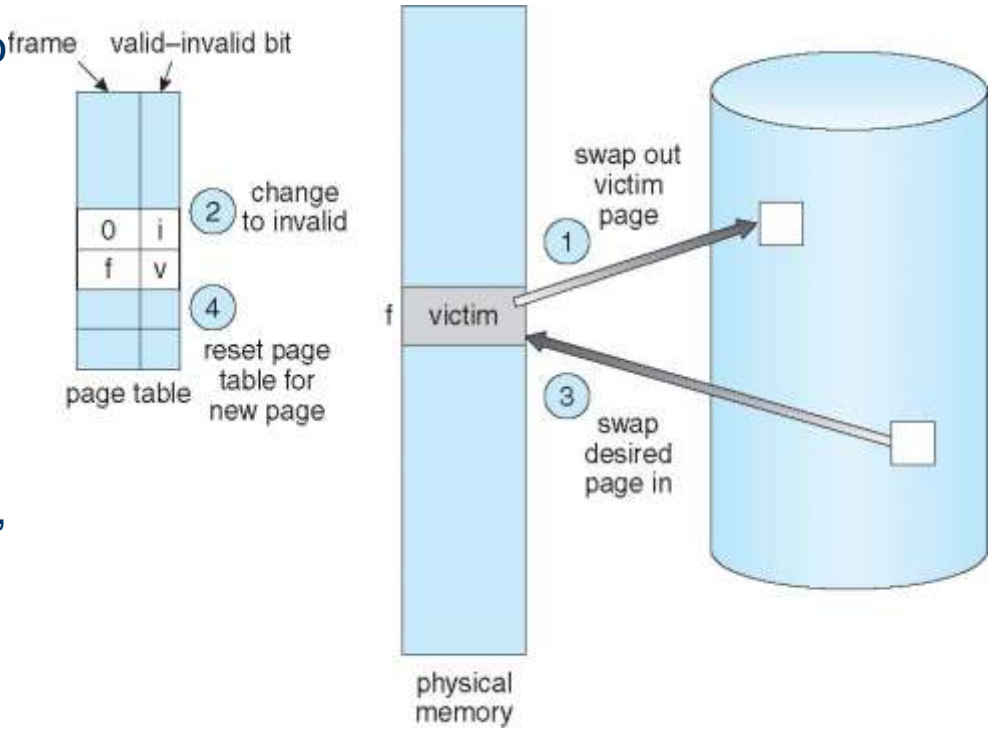
Potrzeba zastępowania stron - przykład



Podstawowy algorytm wymiany stron

1. Znajdź położenie potrzebnej strony na dysku (urządzeniu wymiany)
2. Znajdź wolną ramkę.
   - Jeśli brak wolnej ramki użyj algorytmu zastępowania stron do znalezienia ramki-ofiary (*victim frame*). Zapisz zawartość ramki-ofiary na dysk (jeśli potrzeba) i zwolnij ramkę
3. Wczytaj potrzebną stronę do wolnej ramki, zaktualizuj tablicę stron.
4. Wznów proces restartując instrukcję, która wywołała błąd braku strony

Uwaga: błąd braku strony może powodować aż **2 przesłania stron**, zwiększając EAT.

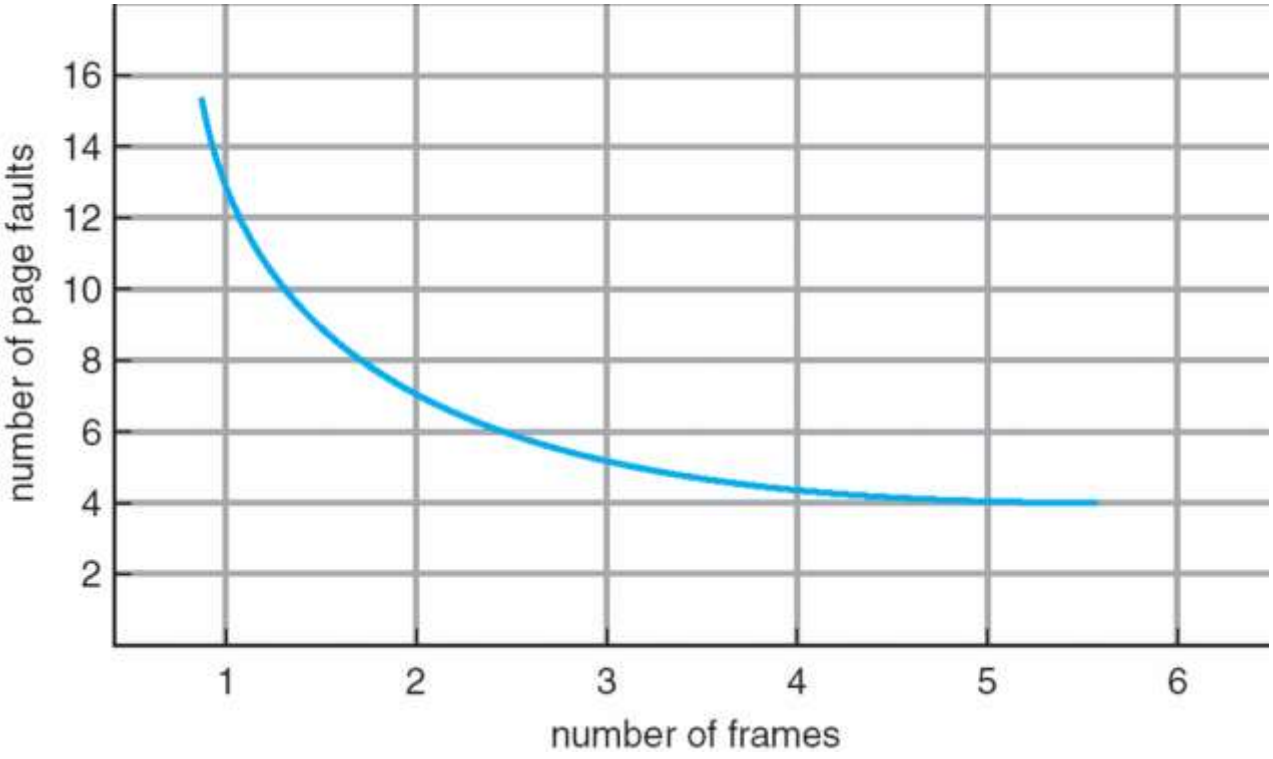


Algorytmy zastępowania stron i ramek

- **Algorytm przydziału ramek (frame-allocation algorithm)** określa
  - liczbę ramek przydzielonych procesowi
  - które ramki mają być zastąpione nową treścią
- **Algorytm zastępowania stron (page-replacement algorithm)**
  - powinien zapewniać małą częstość braków stron; zarówno przy pierwszym odniesieniu do strony jak i przy następnych odniesieniach
- Ocena algorytmu - wyznaczanie liczby braków stron dla danego *ciągu odniesień* do pamięci.
  - Ciąg odniesień, to ciąg numerów stron – a nie pełnych adresów
  - Powtórzenie odniesienia nie powoduje błędu braku strony
  - Wyniki oceny zależą od liczby dostępnych ramek
- W przykładach użyto ciągu odniesień:

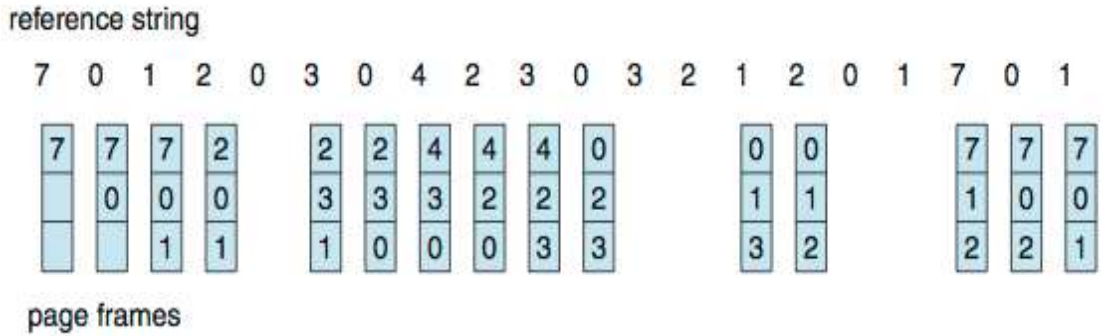
7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Oczekiwana zależność liczby błędów braku strony od liczby ramek



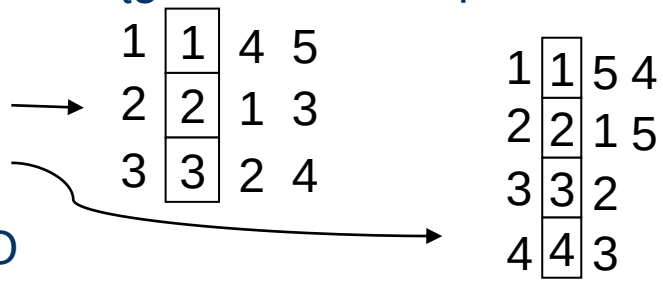
Algorytm FIFO

- Ciąg odniesień: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 ramki (3 strony mogą być jednocześnie w pamięci)



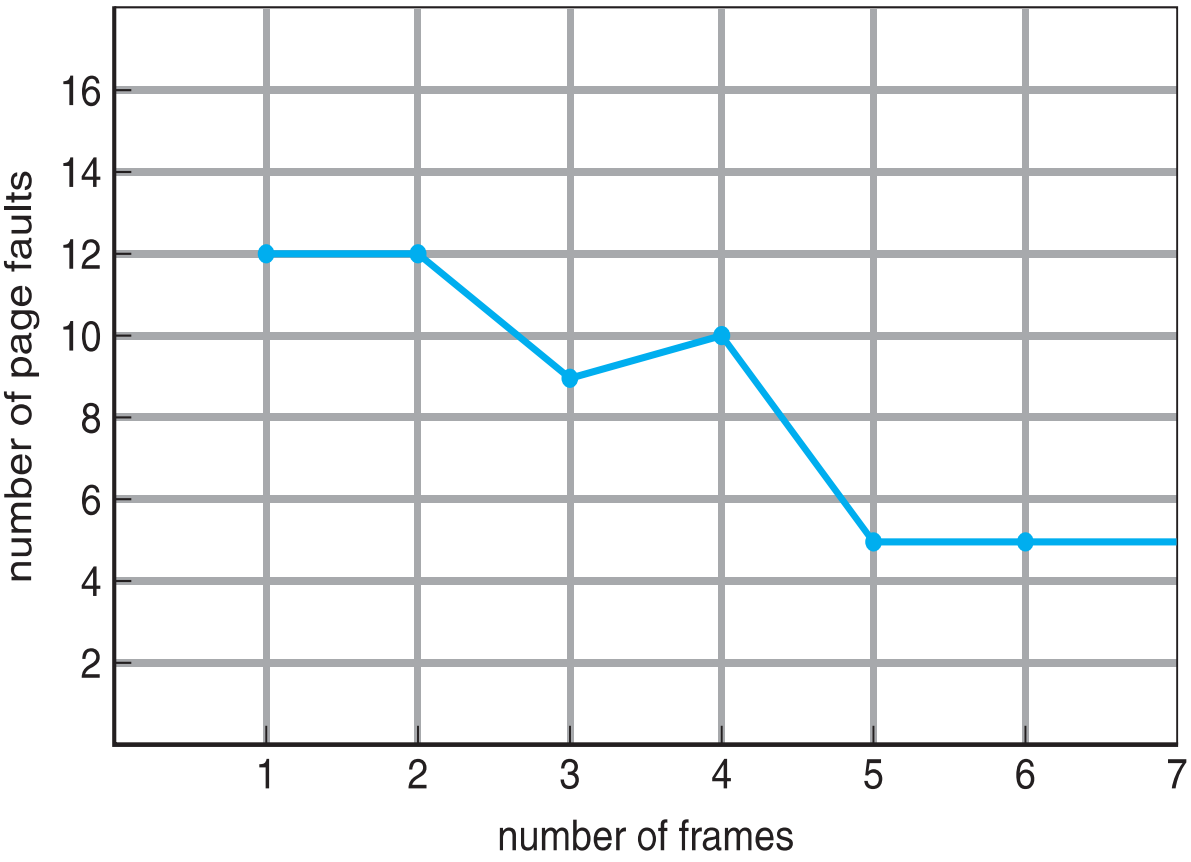
15 błędów braku strony

- 4 ramki -> 10 błędów braku strony
- Liczba błędów braku stron zależy silnie od ciągu odniesień, np. dla ciągu: **1,2,3,4,1,2,5,1,2,3,4,5**
  - 3 ramki -> 9 błędów braku strony
  - 4 ramki -> 10 błędów braku strony



- Anomalia Belady'ego w algorytmie FIFO
  - więcej dostępnych ramek nie musi oznaczać mniej braków stron
- Jak śledzić „wiek strony"
  - Wystarczy kolejka FIFO

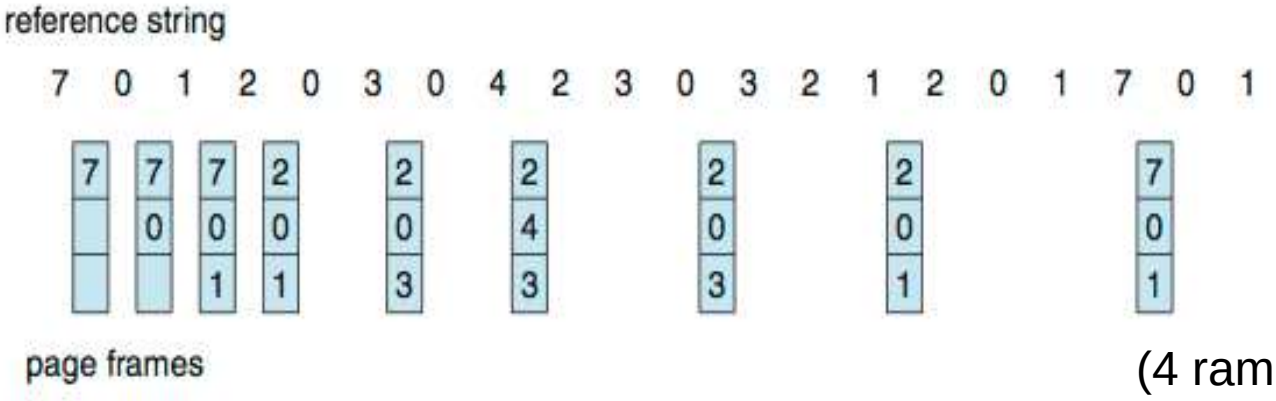
Ilustracja efektu Belady'ego



Ciąg odniesień: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Algorytm optymalny (OPT)

- Zastąp tę stronę, która najdłużej nie będzie używana.
- Przykład z 3 ramkami i ciągiem odniesień:
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

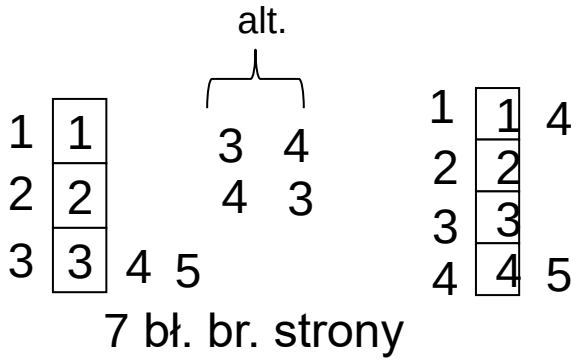


3 ramki – 9 błędów braku strony

(4 ramki – 8 bł. braku strony)

- Przykład z ciągiem odniesień

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

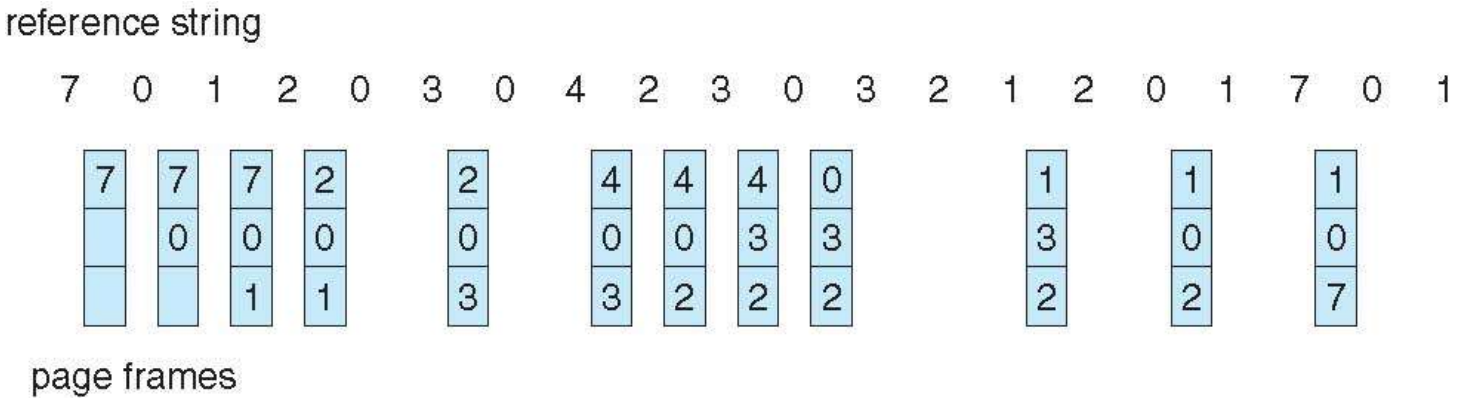


6 bł. braku strony

- Jak poznać przyszłość (odniesień do pamięci)?
- Algorytm OPT (MIN) używany jest głównie w studiach porównawczych.

Algorytm LRU

- LRU używa wiedzy o przeszłości, a nie o przyszłości
- Zamieniana jest strona, która najdłużej nie była używana (*least-recently used page,* LRU)
- Ciąg odniesień: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**



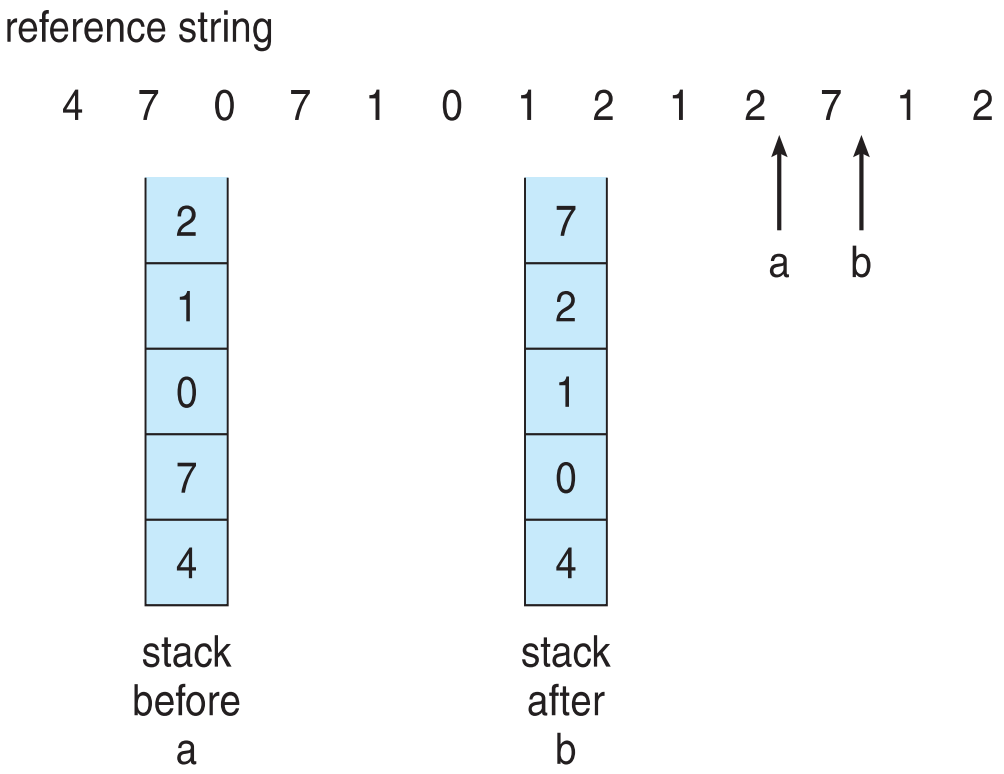
- 3 ramki - 12 błędów braku stron (FIFO: 15, OPT: 9)
- Algorytmy stosowe LRU i OPT nie wykazują efektu Belady'ego
- Ogólnie dobry, często stosowany algorytm.
- Jak implementować LRU?

Algorytm LRU (c.d.)

- Implementacja z licznikiem
  - Każda strona ma rejestr czasu użycia; przy odniesieniu do strony rejestr zegara jest kopiowany do rejestru życia strony.
  - Zastępowana jest strona z najmniejszym znacznikiem czasu.

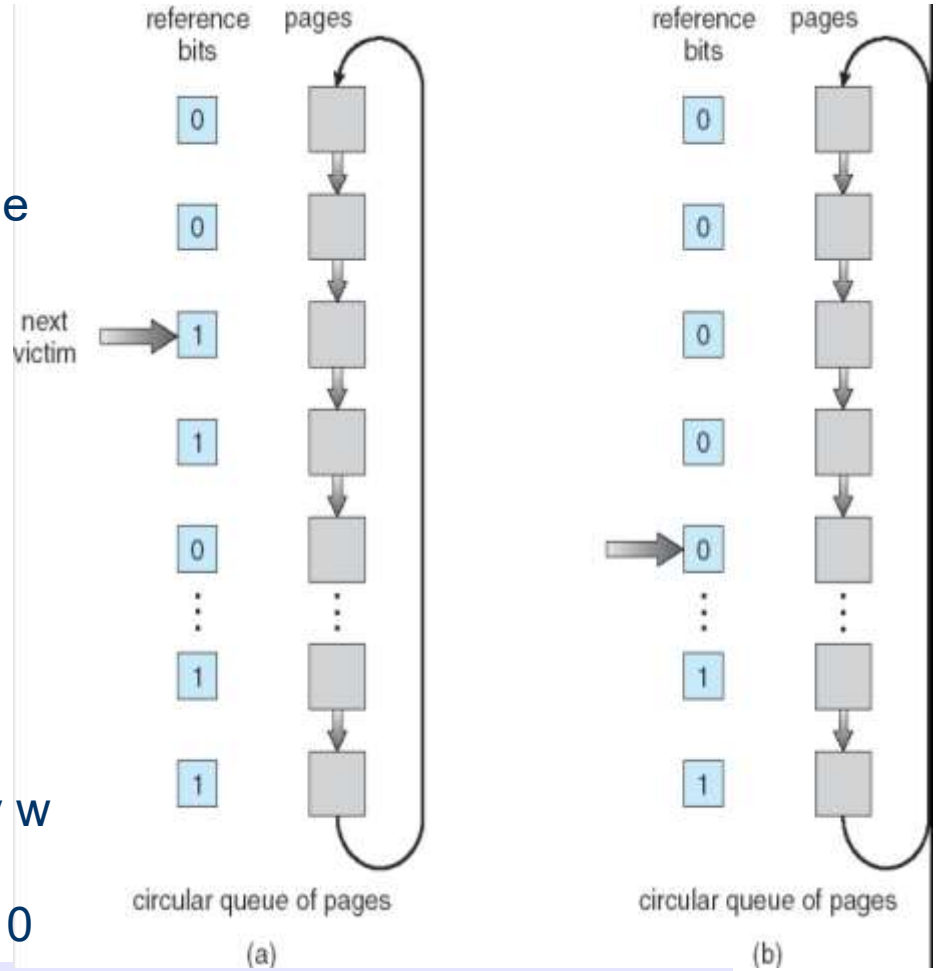
- Implementacja stosowa przechowuje stos numerów stron w dwukierunkowej liście:
  - odniesienie do strony powoduje przeniesienie numeru strony na szczyt stosu
  - brak poszukiwania ofiary (ofiara na dole stosu)
  - Aktualizacja stosu – bardziej kosztowna niż licznika

Wykorzystanie stosu do realizacji algorytmu LRU



Algorytmy przybliżające LRU

- Bit odniesienia
  - z każdą stroną związany jest bit odniesienia, początkowo = 0
  - odniesienie do strony ustawia bit odniesienia na 1.
  - Można zastąpić strony z bitem odniesienia 0 (jeśli takie istnieją); nie wiadomo w jakiej kolejności.
- Algorytm drugiej szansy ➔
  - Potrzebny jest bit odniesienia.
  - Zastępowanie z użyciem cyklicznej kolejki stron (alg. zegarowy):
    - Jeśli bit odniesienia = 1, to:
      - zmień bit odniesienia na 0, pozostawiając stronę w pamięci
      - przejdź do następnej strony w kolejce.
    - Usuń stronę z bitem odniesienia = 0



Ulepszony algorytm drugiej szansy

- Algorytm używa bitu odniesienia i dodatkowego bitu modyfikacji
- Dla pary bitów (odniesienia, modyfikacji)

1. (0, 0) nie było ostatnio modyfikacji ani odniesienia – najlepsza strona do usunięcia
2. (0, 1) modyfikowana, ale ostatnio nie używana – drugi wybór, bo trzeba zapisać stronę do pamięci wymiany przed jej usunięciem
3. (1, 0) ostatnio używana i nie modyfikowana – prawdopodobnie będzie wkrótce ponownie użyta
4. (1, 1) ostatnio używana i modyfikowana – prawdopodobnie będzie wkrótce ponownie użyta; usunięcie możliwe po zapamiętaniu.

- Kiedy konieczna jest wymiana stron - w algorytmie zegarowym warto szukać strony w klasie 1, a jeśli ta klasa jest pusta w klasie 2 itd. Jest prawdopodobne, że kolejkę okrężną trzeba obiegać kilka razy, by znaleźć właściwą ofiarę.

Algorytmy zliczające

- Można wprowadzić liczniki odniesień do każdej ze stron i postępować na jeden z dwóch poniższych sposobów:

- Algorytm LFU: zastąp stronę z najmniejszą wartością licznika. Uwaga: strony dopiero co wprowadzone są zagrożone usunięciem.

- Algorytm MFU: zastąp stronę z największą wartością licznika

Buforowanie stron

- System operacyjny zawsze utrzymuje zbiór wolnych ramek
  - Ramka jest dostępna natychmiast gdy będzie potrzebna do obsługi błędu braku strony
  - Po wczytaniu strony do wolnej ramki system wybiera proces-ofiarę, któremu odbierze ramkę, przekazując ją do zbioru ramek wolnych
  - Kiedy jest to wygodne system może odebrać ofierze wszystkie ramki
- Jeśli to możliwe system utrzymuje listę stron zmodyfikowanych
  - W czasie, gdy pamięć wymiany jest wolna system zapisuje zawartość stron zmodyfikowanych, zmieniając bit modyfikacji (*non-dirty, clean*).
- Jeśli to możliwe system powinien możliwie długo powstrzymywać czyszczenie ramek ze zbioru wolnych ramek
  - Jeśli pojawi się zlecenie użycia strony zawartej w ramce wolnej – można natychmiast wykorzystać ramkę po usunięciu ze zbioru ramek wolnych (*soft page fault*).
  - Dobrze jest minimalizować negatywne skutki złego wyboru ramki odbieranej procesie-ofierze.

Wymiana stron a oczekiwania aplikacji względem dostępu do pamięci

- Omówione algorytmy próbują zgadywać przyszłe wykorzystanie pamięci
- Niektóre programy użytkowe, np. bazy danych, mogą mieć znacznie lepszą wiedzę o wzorcach dostępu do pamięci
- Aplikacje wykorzystujące intensywnie pamięć tracą dużo na efektywności wskutek podwójnego buforowania
  - SO przechowuje kopię danych w buforze wejścia/wyjścia
  - Aplikacja kopiuje dane do swojej pamięci (bufor użytkownika)
- System operacyjny może udostępnić dostęp do dysku w trybie surowym (***raw disk*** *mode*), co likwiduje w/w problemy

Przydział ramek

- Każdy proces potrzebuje pewnego minimum liczby stron.
- Przykład: IBM 370 – potrzebuje 6 stron dla obsłużenia rozkazu przemieszczania znaków (w najgorszym przypadku):
  - rozkaz zajmuje 6 bajtów, może wymagać 2 stron,
  - 2 strony może zajmować blok znaków do przesłania (**from**)
  - 2 strony może zajmować obszar docelowy (**to)**.
- Dwa podstawowe typy algorytmów przydziału:
  - przydział równy
  - przydział priorytetowy (np. proporcjonalny)

Przydział równy

- Przydział równy – np. dla 100 ramek i 5 procesów każdy proces otrzymuje po 20 ramek..
- Przydział proporcjonalny do rozmiaru procesu .

$$s_i = \text{rozmiar procesu } p_i$$

$$S = \sum s_i$$

$$m = \text{całkowita liczba ramek}$$

$$a_i = \text{przydział dla procesu } p_i := \frac{s_i}{S} \times m$$

$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Przydział priorytetowy

- Proces otrzymuje przydział pamięci proporcjonalnie do jego priorytetu, a nie rozmiaru.

- Jeśli proces P_i spowodował błąd braku strony, to można:
  - zastąpić jedną z ramek tego procesu.
  - wybrać do zastąpienia ramkę innego procesu o niższym priorytecie.

Przydział globalny a lokalny

- **Przydział globalny** – proces może “podkradać” ramki innemu, zwiększając liczbę przydzielonych sobie ramek. Proces nie może kontrolować własnej częstości braków stron (która zależy też od stronicowania innych procesów).

- **Przydział lokalny** – proces zastępuje ramki przydzielone mu przez system. Zbiór stron procesów w pamięci zależy tylko od stronicowania odnoszącego się tylko do danego pamięci. Niezmienna liczba przydzielonych ramek może hamować proces.

Niejednorodny dostęp do pamięci (NUMA)

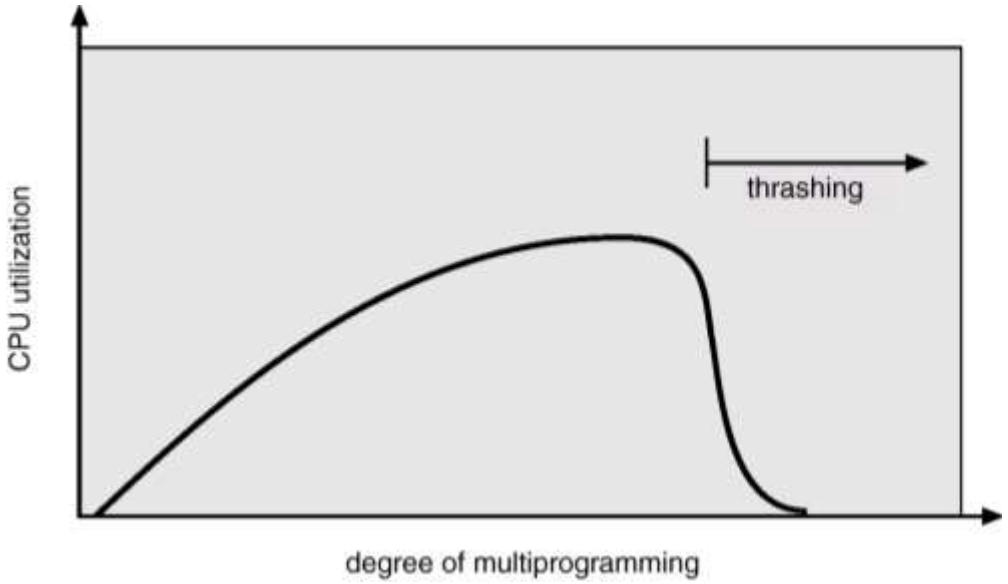
- Do tej pory rozważaliśmy jednolity dostęp do pamięci fizycznej
- Wiele systemów posiada niejednorodny dostęp do pamięci (**NUMA**) – czas dostępu jest zmienny
  - Przykład: system zawiera płyty, połączone szybkimi kanałami komunikacyjnymi; każda płyta zawiera procesor i pamięć RAM.
- Największą wydajność dostępu do pamięci uzyskuje się wykorzystując jednostki pamięci „bliskie” procesorowi/rdzeniowi, który wykonuje dany wątek procesu.
  - Takie zachowanie wymaga odpowiedniej budowy planisty przydziału procesorów.
  - W interfejsie systemu NUMA istnieje możliwość podpowiadania właściwego przydziału procesora (*processor affinity of a process, thread*)

Szamotanie (thrashing)

- Jeśli proces nie ma przydzielonej dostatecznie dużej liczby ramek duża jest częstość braku stron. Konsekwencje:
  - małe wykorzystanie procesora.
  - system operacyjny próbuje poprawić wykorzystanie procesora zwiększając stopień wieloprogramowości, tzn. wprowadza nowe procesy.
  - nowe procesy wymagają wolnej pamięci

- **Szamotanie** ≡ proces spędza więcej czasu oczekując na wymianę stron niż na wykonanie.

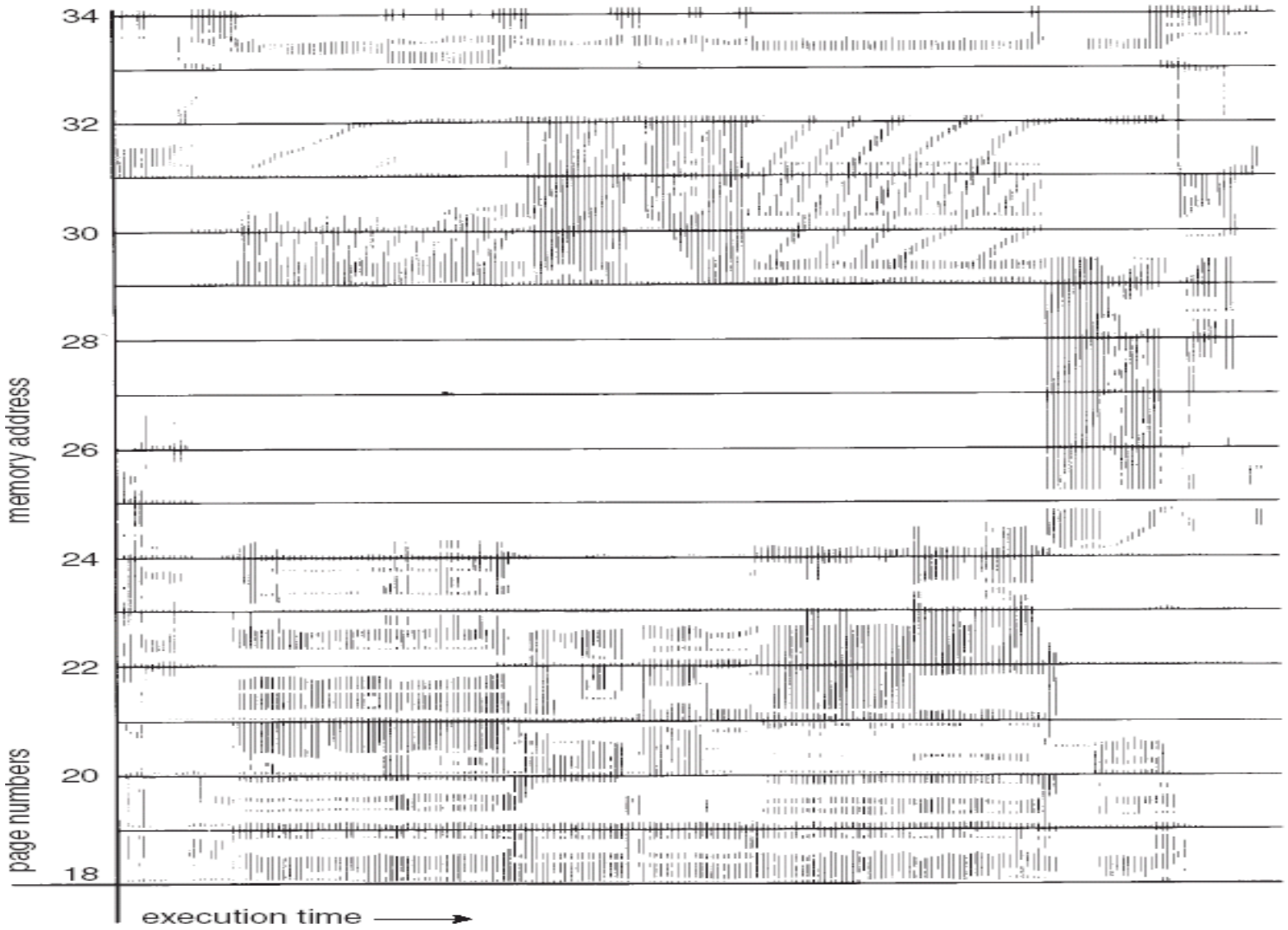
Szamotanie



- Dlaczego stronicowanie jest przydatne?
  Model strefowy (locality model)
  - Proces w trakcie wykonania przechodzi z jednej strefy do innej.
    Strefa = zbiór stron pozostających we wspólnym użyciu.
  - Strefy mogą mieć części wspólne.
- Dlaczego występuje szamotanie?

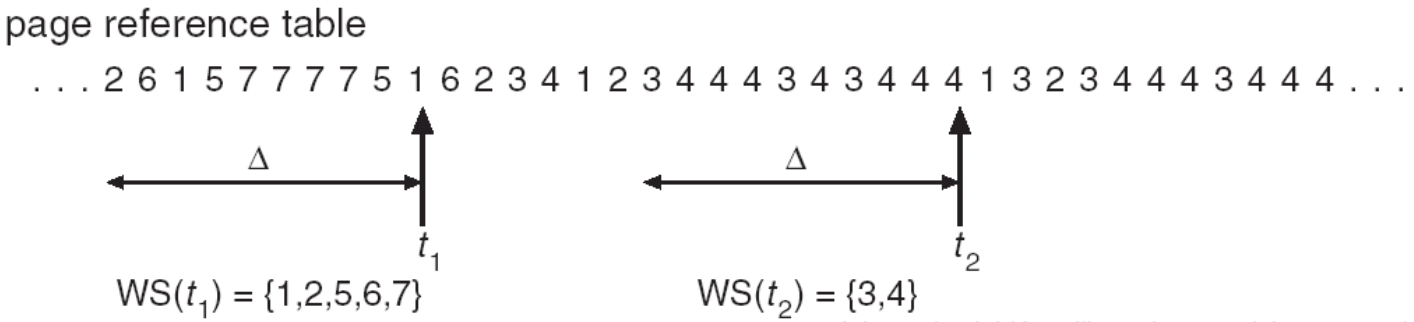
  Σ rozmiarów stref > całkowity rozmiar pamięci
- Ograniczenie skutków: zastępowanie lokalne, albo priorytetowe

Lokalność odniesień do pamięci



Model zbioru roboczego

- $\Delta \equiv$ okno zbioru roboczego $\equiv$ ustalona liczba odniesień do stron pamięci, np. 10,000 instrukcji
- WS_i (zbiór roboczy, *working set of process* P_i) =
  zbiór stron do których były odniesienia w ostatnim oknie Δ (rozmiar: WSS_i)
  - jeśli Δ jest zbyt małe, to WSS_i nie obejmuje całej strefy procesu
  - jeśli Δ jest zbyt duże, to WSS_i będzie obejmować kilka stref procesu
  - jeśli $\Delta = \infty \Rightarrow$, to WSS_i będzie obejmować cały proces.
- $D = \Sigma\ WSS_i \equiv$ całkowite zapotrzebowanie na ramki
- if $D > m \Rightarrow$ szamotanie (*thrashing*)
- Polityka: jeśli $D >$ m, to trzeba uśpić jeden z procesów.



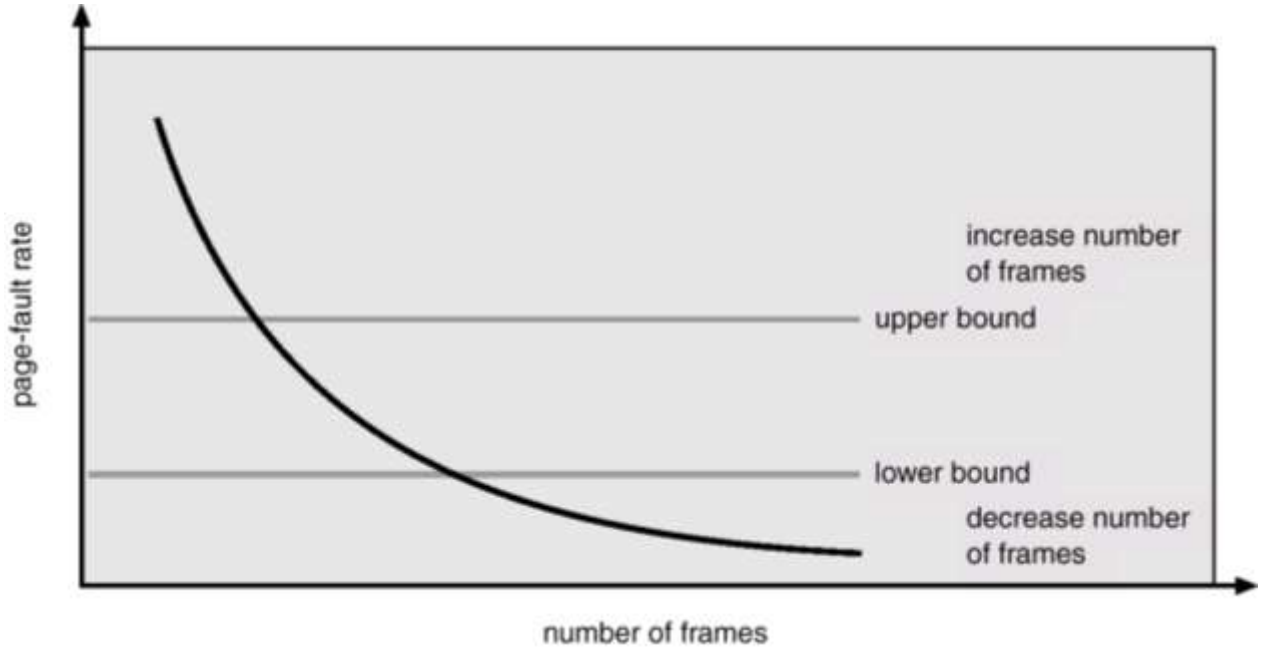
Tworzenie modelu zbioru roboczego

- Aproksymacja za pomocą regularnych przerwań zegarowych i bitu odniesienia

- Przykład: Δ = 10,000
  - Przerwania zegarowe co 5000 jednostek czasu.
  - Dla każdej strony jest dodatkowy przesuwny rejestr 2 bitowy.
  - Dla każdego przerwania zegarowego wartości bitu odniesień stron są kopiowane do rejestru, a bity odniesienia są zerowane.
  - Jeśli jeden z bitów rejestru = 1 $\Rightarrow$ strona należy do zbioru roboczego.

- Dlaczego nie jest to metoda dokładna?
- Ulepszenie: 10 bitowy rejestr i przerwanie co 1000 jednostek czasu.

Częstość braku stron



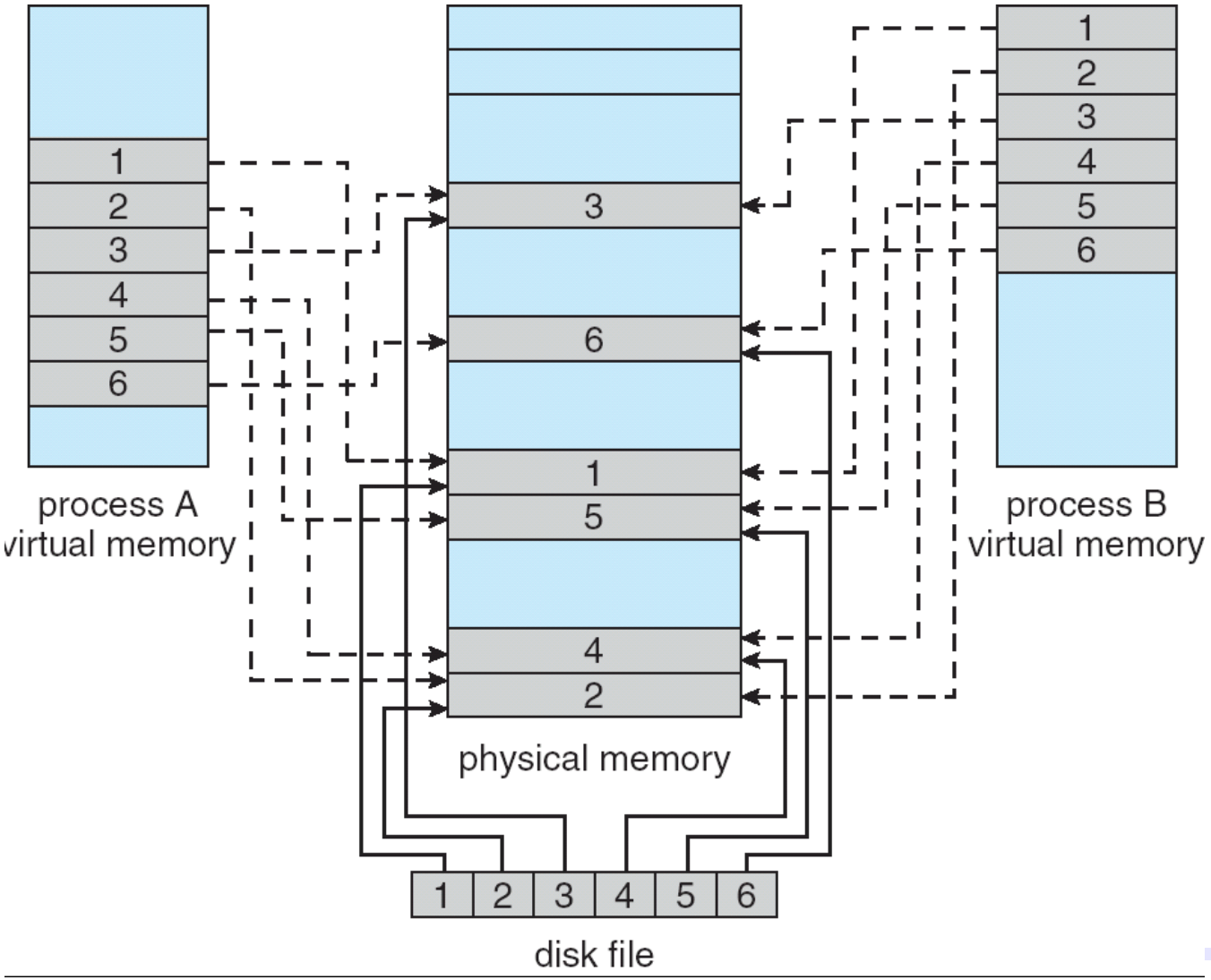
Chodzi o osiągnięcie “akceptowalnej” częstości braku stron:

- za duża wartość świadczy o tym, że proces potrzebuje więcej ramek, a
- za mała wartość może oznaczać, że proces ma zbyt wiele ramek.

Odwzorowanie plików w pamięci

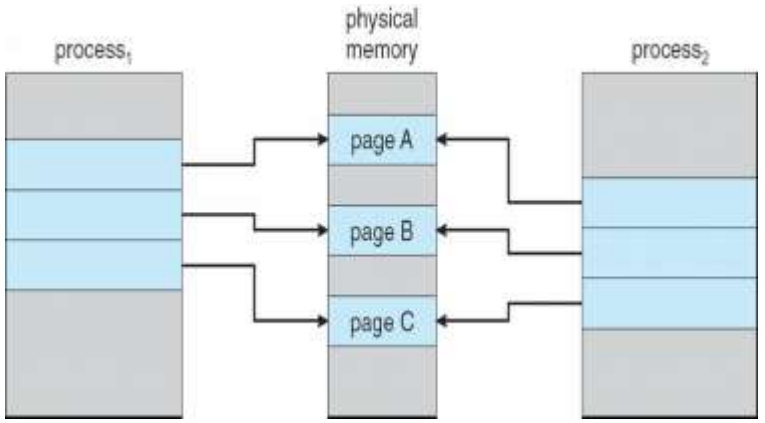
- Odwzorowywanie plików w pamięci (*memory-mapped file I/O)* polega na logicznym skojarzeniu z plikiem (bądź jego częścią) fragmentu wirtualnej przestrzeni adresowej.
- Pierwszy dostęp do pliku jest realizowany za pomocą stronicowania na żądanie – skutkuje błędem braku strony. Następne operacje czytania i zapisu pliku będą realizowane jak zwyczajne dostępy do pamięci operacyjnej (być może po kolejnej obsłudze błędu braku strony).
- Dostęp do pliku jest uproszczony – operacje na tablicy zamiast wywoływania funkcji systemowych (**`read()`**, **`write()`**)
- Zapisy do pamięci operacyjnej nie oznaczają koniecznie zapisu danych do pliku. Synchronizacja: na żądanie, okresowa bądź przy zamykaniu pliku.
- Istnieje możliwość odwzorowywania tego samego pliku przez więcej niż jeden proces (pamięć dzielona).
- W niektórych systemach standardowe wejście/wyjście jest realizowane za pomocą odwzorowania plików w pamięci.

Odwzorowanie plików w pamięci

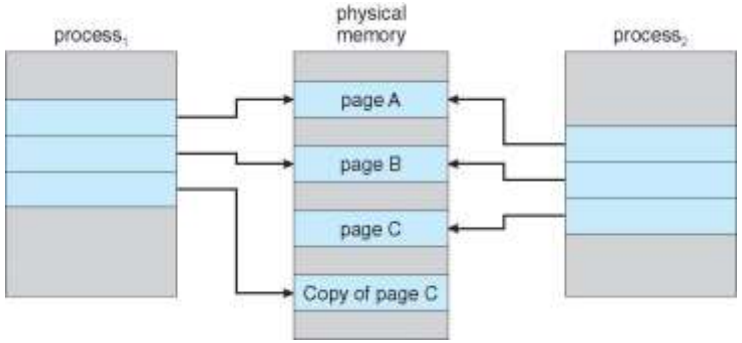


Kopiowanie przy zapisie

- **Kopiowanie przy zapisie** (Copy-on-Write, COW) pozwala na współdzielenie fragmentów przestrzeni adresowych przez dwa lub więcej procesów. Procesowi, który próbuje zmodyfikować współdzieloną pamięć, przydziela się prywatną kopię modyfikowanej strony i odniesienie do pamięci jest przekierowane.



- Dzięki COW proces potomny utworzony funkcją fork() może zaraz po utworzeniu dzielić większość przestrzeni adresowej z procesem macierzystym. Dzięki temu realizacja COW funkcji fork() jest bardziej efektywna od „tradycyjnej”/naiwnej.



Przydział pamięci jądra

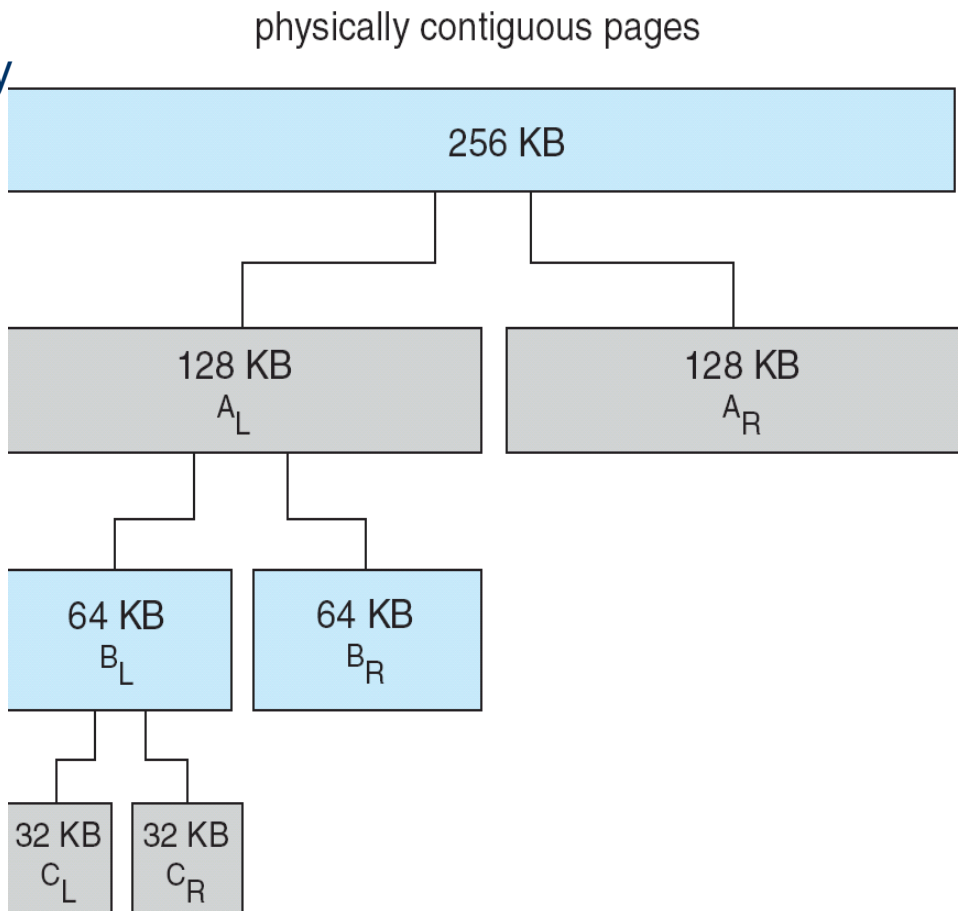
- Przydział pamięci dla podsystemów jądra jest zorganizowany inaczej niż przydział pamięci dla procesów użytkownika
  - Jądro potrzebuje porcji pamięci różnej wielkości, zazwyczaj na krótki czas
  - w większości zleceń rozmiar zamawianej pamięci jest znacznie mniejszy od rozmiaru strony

- Kryteria oceny modułu przydzielającego pamięć jądra:
  - zdolność do minimalizowania strat (fragmentacja wewnętrzna i zewnętrzna)
  - szybkość działania

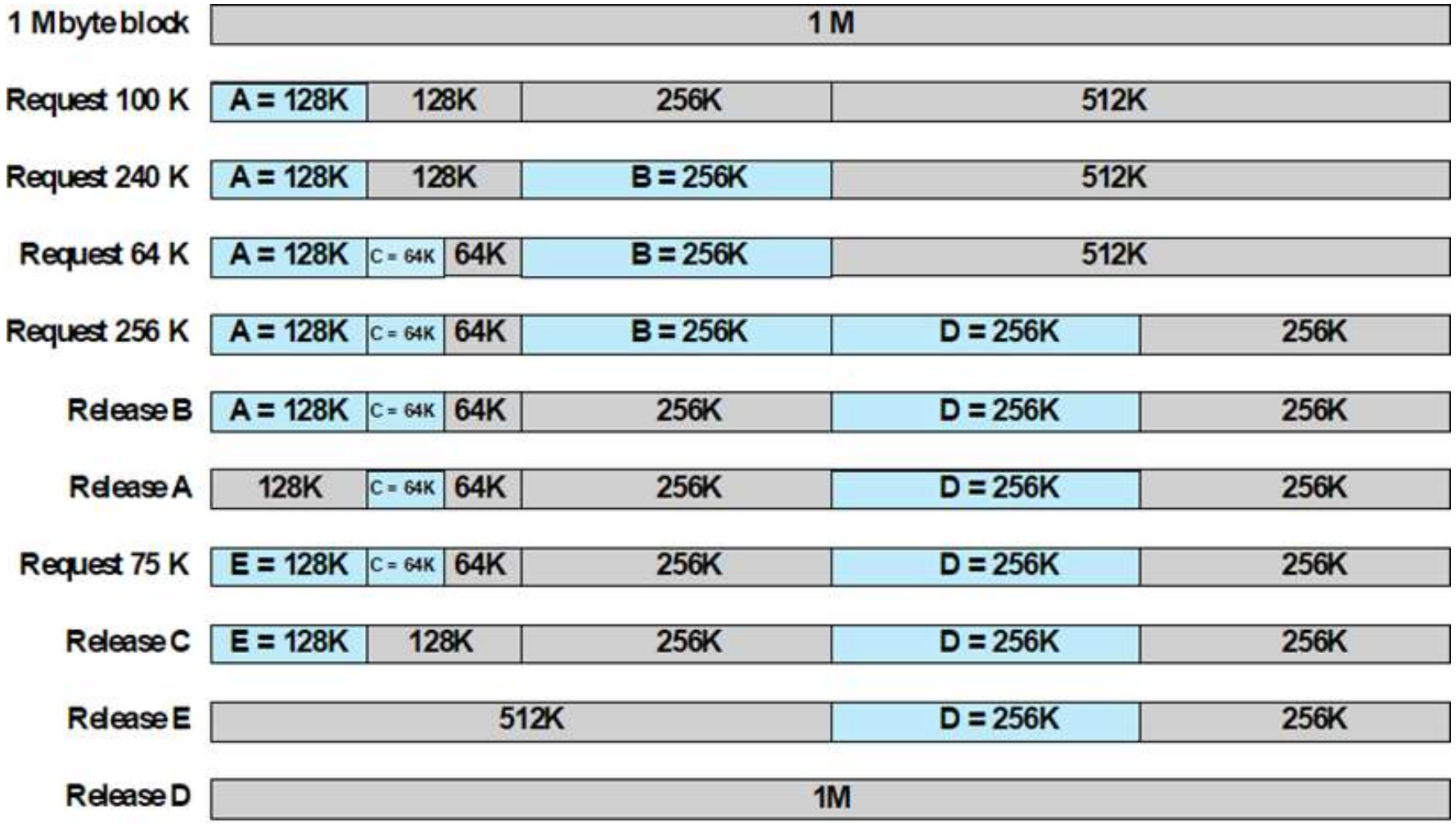
- Popularne sposoby organizacji przydziału pamięci jądra:
  - system bliźniaków (*buddy system*)
  - płytkowy moduł przydziału (*slab*)

Algorytm bliźniaków

- Przydział obszarów pamięci ze spójnego zbioru stron.
- Zamówienie na pamięć zaokrąglone do wielkości 2^n. Jeśli nie ma fragmentu o tej wielkości większy obszar jest dzielony na połowy itd. – aż do uzyskania najmniejszego bloku spełniającego wymagania.
- Zaleta: szybkość scalania nieużywanych fragmentów
- Wada: fragmentacja pamięci



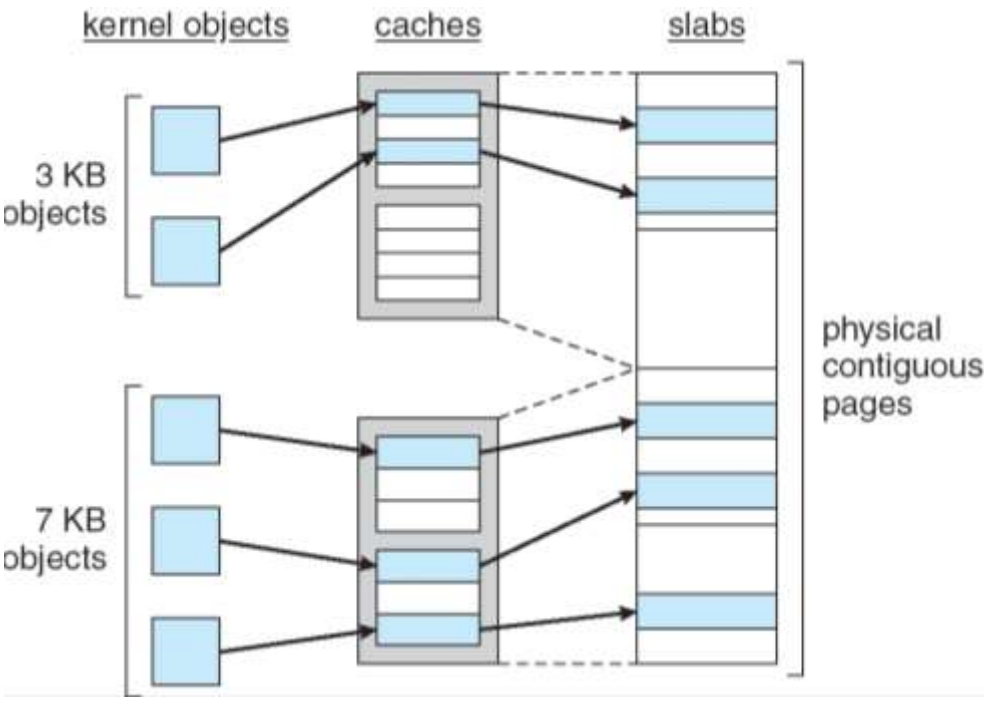
Przykład wykorzystania bliźniaczego przydziału pamięci



Informacja o stanie przydziałów może być przechowywana w drzewie binarnym.

Płytkowy moduł przydzielający

- Struktura Cache zawiera obiekty jądra tego samego typu/rozmiaru.
- Pamięć struktury pochodzi z jednego lub więcej bloków „płyty” pamięci (slab)
- Nowo utworzona struktura Cache zawiera pulę pustych obiektów (znacznik **`free`**). Zapis nowej treści obiektu zmienia jej znacznik na **`used`**
- Jeśli struktura Cache zapełniła blok pamięci płyty (slab) przydzielony jest pusty (istniejący lub nowy blok).
- Zalety: brak fragmentacji, szybka obsługa zlecenia przydziału pamięci (dzięki pre-alokacji).



Alokator płytkowy w SO Linux

- Alokatory płytkowe, wprowadzone w SO Solaris, są obecnie szeroko stosowane w SO
- Linux 2.2 miał podstawowy wariant alokatora: SLAB, obecnie ma również SLOB i SLUB
  - SLOB (Simple List of Blocks) stosowany dla systemów z ograniczoną pamięcią. Utrzymuje 3 listy obiektów: małych, średniej wielkości oraz dużych. Ma problem z fragmentacją. Poszukuje bloku pamięci za pomocą algorytmu first-fit.
  - SLUB (the unqueued slab allocator) jest wersją SLAB o powiększonej wydajności (usunięto kolejki związane z procesorami), usunięto problem fragmentacji
- Przykładowo struktura procesu, typu `struct task_struct;` zajmuje ok. 1.7KB
- Tworzenie nowego zadania -> przydział wolnej struktury z puli „Cache”
- Płytka (slab) może znajdować się w jednym z trzech stanów:
  1. Full – brak wolnych
  2. Empty – brak przydziałów
  3. Partial – częściowo pusta/pełna
- Alokator płytkowy może (na zlecenie):
  1. Wykorzystać wolną strukturę w płytce o stanie “partial slab”
  2. W przeciwnym przypadku wykorzystuje strukturę płytki o stanie “empty slab”
  3. A jeśli to niemożliwe – tworzy nową płytkę o stanie “ empty slab“.

Inne rozważania

- **Stronicowanie wstępne** (*pre-paging*)
  - Jest stosowane, by wyeliminować dużą liczbę błędów braku strony, które mogłyby się pojawić przy starcie procesu.
  - Polega na wprowadzeniu do pamięci fizycznej wszystkich stron, albo tylko wybranych stron, które proces będzie używał, ale zanim proces wygeneruje stosowne odniesienia do pamięci.
  - Jeśli zawczasu wprowadzone strony nie będą jednak przez proces użyte – zajętość pamięci i praca urządzeń wejścia/wyjścia pójdą na straty
- **Zasięg TLB** – pamięć dostępna za pomocą (wpisów) tego rejestru

  TLB Reach = (TLB Size) X (Page Size)
  - Idealnie cały zbiór roboczy powinien być osiągalny z TLB
  - Zwiększanie zasięgu przez wzrost rozmiaru strony prowadzi do zwiększenia fragmentacji wewnętrznej
  - Dobry kompromis uzyskuje się w systemach, w których dostępne jest jednocześnie **kilka rozmiarów stron**

Inne rozważania (c.d.)

- Struktura programu wpływa na wykorzystanie pamięci wirtualnej:
  - Tablica `int data [1024][1024];`
  - Niech każdy wiersz tablicy będzie pamiętany na oddzielnej stronie

  - Program 1

```
for (j = 0; j <1024; j++)
for (i = 0; i <1024; i++)
     data[i][j] = 0;
```

    wytwarza nawet do 1024 x 1024 błędów braku strony

  - Program 2

```
for (i = 0; i < 1024; i++)
  for (j = 0; j < 1024; j++)
  data[i][j] = 0;
```

    wytwarza 1024 błędów braku strony

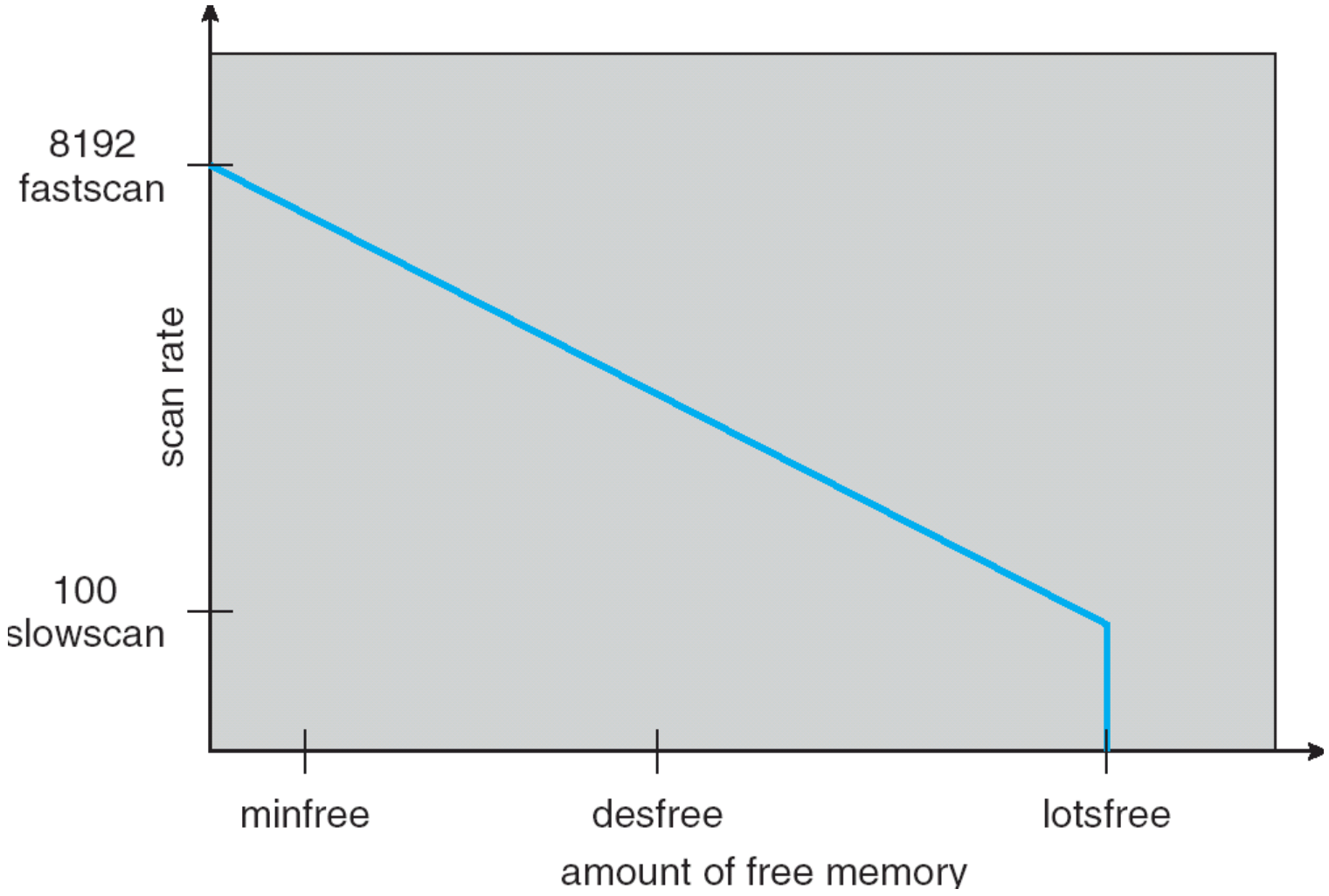
Przykłady realizacji VM: Windows

- Stronicowanie na żądanie wykorzystuje **grupowanie stron** (*clustering*), tzn. sprowadzona jest nie tylko strona brakująca, ale i pewne jej otoczenie.
- Proces po utworzeniu ma określone **minimum i maksimum rozmiaru zbioru roboczego**. Proces ma zagwarantowany przydział minimum, ale może mieć przydzielone aż do deklarowanego maksimum. Dalsze próby przydziału są realizowane przez (lokalne) zastępowanie stron.
- Jeśli rozmiar wolnej pamięci w systemie spada poniżej pewnej wartości, a procesowi przydzielono więcej stron niż minimum zbioru roboczego, to zarządca pamięci wirtualnej stosuje automatyczne przycinanie (*automatic working set trimming*) – próbując odbierać procesowi nadmiarowe strony (np. posługując się odmianą algorytmu zegarowego)
- Zarządca pamięci wirtualnej przydziela procesowi pamięć w dwóch krokach:
  1. Rezerwuje część przestrzeni wirtualnej procesu (*reserved memory*)
  2. Przydziela zarezerwowanemu obszarowi pamięć fizyczną oraz pamięć na urządzeniu wymiany (*commited memory*)

Przykłady realizacji VM: UNIX

- Zarządca VM utrzymuje listę wolnych stron, aby móc szybko przydzielać pamięć procesom generującym błędy braku strony.
- 4 razy/sekundę jądro sprawdza, czy ilość wolnych stron >= *Lotsfree.* Jeśli nie jest - uruchomiony jest proces (*pageout*) który realizuje „wymiatanie stron”. Chodzi o przenoszenie stron nieaktywnych na listę stron do zwolnienia, po zapisie modyfikacji tych stron. Strony mogą jeszcze powrócić do zbioru aktywnych – jeśli nie zostaną nadpisane nową zawartością.
- W algorytmie wymiatania stron używa się algorytmu zegara dwuwskazówkowego. *Scanrate* to szybkość analizowania stron: waha się od *slowscan* do *fastscan.* Odległość pomiędzy wskazówkami (*handspread*) i wartość *scanrate* zależą od ilości wolnej pamięci.
- Jeżeli wolnych stron jest mniej niż *Desfree* – *pageout* uruchamiany jest 100 razy/sekundę. Jeśli nie uda się zapewnić w ciągu 30s przynajmniej *Desfree* wolnych stron - jądro rozpoczyna wymianę procesów (*swapping*).
- Jeżeli system nie potrafi utrzymać *Minfree* stron wolnej pamięci – proces wymiatania stron jest wywoływany przy zamówieniu każdej nowej strony

Analizator stron w systemie Solaris 2

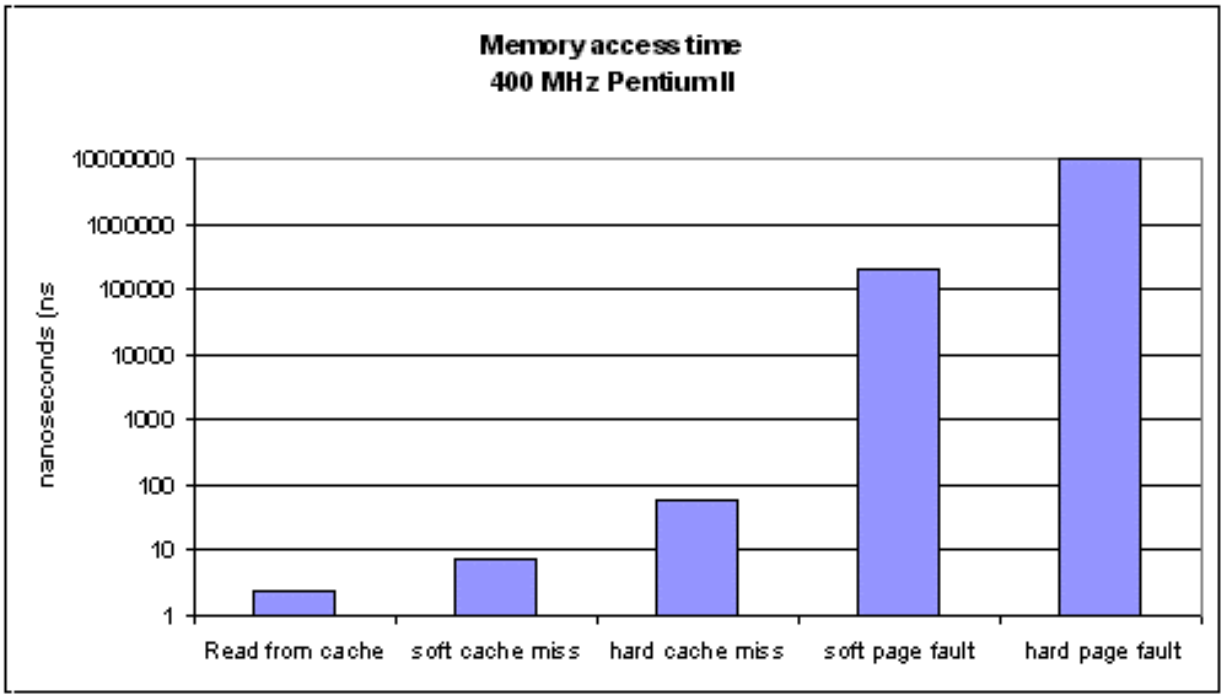


Linux

- Pamięć wirtualna realizowana jest przez stronicowanie na żądanie
- Rozmiar strony: 4KB (Intel Pentium, Intel 64), 8KB (DEC Alpha), 4KB/ 8MB (Intel 64)
- Algorytm przydziału (alokator) stron wykorzystuje algorytm bliźniaków. Jądro przechowuje listę ciągłych grup ramek o stałym rozmiarze (1,2,4,8,16,32).
- Demon wymiany **kswapd**
  - wątek jądra uruchamiany przez proces init podczas startu systemu
  - budzony w regularnych odstępach czasu, aby sprawdzić, czy liczba wolnych ramek w systemie nie spada poniżej dozwolonego poziomu. Jeśli wolnych ramek jest za mało - demon próbuje:
    - Zmniejszyć rozmiar buforów i cache stron
    - Wytoczyć z pamięci strony współdzielone
    - Wytoczyć lub skasować inne strony

    Algorytm może jednokrotnie pozyskać do 4 stron. Jeśli potrzeba – ponawia pozyskiwanie wolnych ramek.
- Algorytm zastępowania stron: LFU (8 bitowy wiek: dostęp – inkrementacja; okresowa kontrola jądra – dekrementacja; ==0 -> strona stara, do zastąpienia).

Orientacyjny czas dostępu do pamięci PC



- Odczyt bezpośrednio z cache L1 – 1 cykl (2.5 ns)
- Odczyt danej z cache L2 – 3 cykle (7.5 ns)
- Odczyt danej z pamięci operacyjnej – 14 cykli (35 ns)
- Odczyt danej ze strony poza pamięcią – 4000000 cykli (10ms)
- J.w., ale jeśli strona jest buforowana przez drajwer dysku lub jądro – 80000 cykli (200 υs)