Systemy operacyjne
Zarządzanie pamięcią

[2] **Zarządzanie pamięcią**

Czego programiści oczekują od pamięci systemu:

- by była duża,
- by była szybka,
- by była nieulotna.

Hierarchia pamięci:

- mała szybka droga pamięć (np. cache),
- średnia, średnio szybka, umiarkowanie droga, pamięć (np. pamięć operacyjna),
- ogromna, wolna i tania pamięć (np. pamięć dyskowa/ taśmowa).

Zarządzanie pamięcią (ang. *Memory Management, MM*) na poziomie systemu operacyjnego uwarunkowane architekturą systemu.

[3] **Organizacja zarządzania pamięcią**

Na organizację zarządzania mają wpływ:

- pole adresowe argumentów rozkazu,
- miejsce pola adresacji w słowie,
- sprzętowe możliwości przekształcania pola adresacji.

W zależności od długości pola adresacji **przestrzeń adresowa** może pokrywać się z zakresem adresów pamięci operacyjnej, może być większa lub mniejsza.

[4] **Funkcje systemu operacyjnego**

Funkcje systemu operacyjnego w kontekście zarządzania pamięcią:

- zagospodarowanie przestrzeni adresowej poprzez wykorzystanie mechanizmów translacji adresów,
- ochrona zawartości pamięci,

- organizacja dostępu do obszarów dzielonych,
- efektywna organizacja pamięci operacyjnej.

Wielkość pamięci operacyjnej przeznaczonej na kod systemu operacyjnego jest zazwyczaj stała, natomiast bardziej złożony jest przydział pamięci procesom użytkowym.

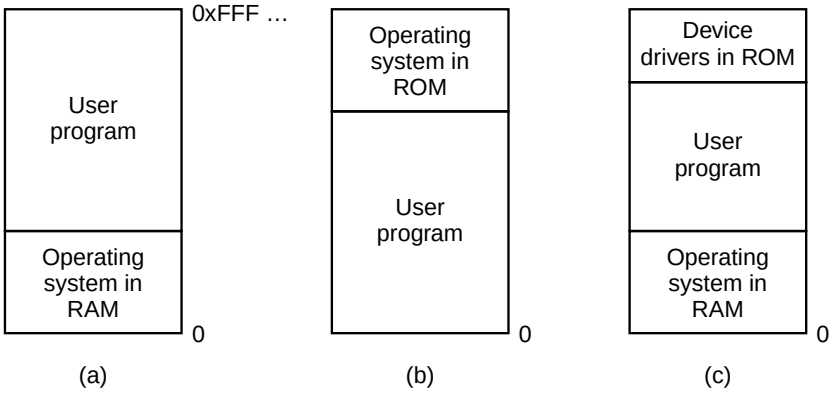
[5] **Metody przydziału pamięci**

Metody przydziału pamięci operacyjnej procesom użytkowym:

1. **brak podziału**, wolna przestrzeń adresowa w danej chwili przydzielana jednemu procesowi użytkowemu. Wieloprogramowanie można realizować przez wymiatanie (ang. *swapping*),
2. **podział pamięci**, wolna przestrzeń adresowa podzielona na części przydzielane pojedynczym procesom użytkowym,
3. wykorzystanie **pamięci wirtualnej**, istnieje jedna lub wiele wirtualnych przestrzeni adresowych przydzielanych procesom użytkowym, a mających w niewielkim stopniu pokrycie w pamięci operacyjnej.

[6] **Brak podziału**

Trzy metody prostej organizacji pamięci dla systemu operacyjnego i jednego procesu uzytkownika.



[7] **Podział pamięci**

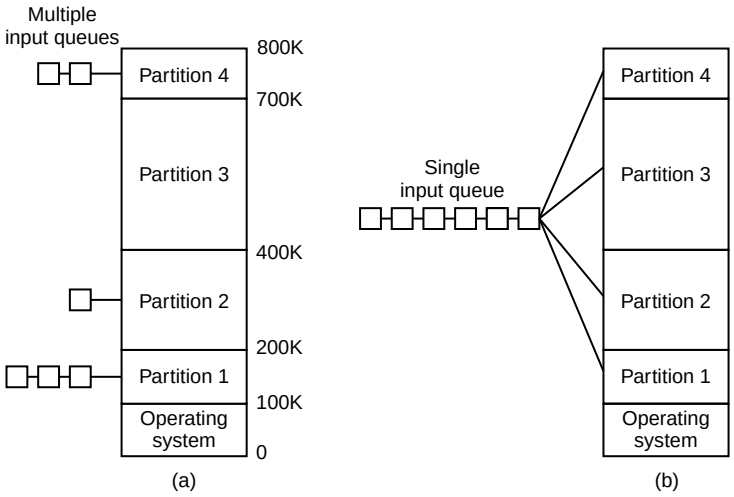
Cele podziału pamięci między procesy:

- lepsze wykorzystanie elementów sprzętowych systemu, głównie procesora i pamięci,
- umożliwienie szybkiego przełączania procesora z pracy z jednym procesem na pracę z innym procesem.

Można wyróżnić następujące typy systemów z podziałem pamięci:

- systemy **z podziałem statycznym**. Podział statyczny dzieli pamięć na stałe partycje o różnej długości lub na bloki o stałej długości zwane **ramami** (ang. *frame*).
- systemy **z podziałem dynamicznym**. Realizacja z wykorzystaniem struktur opisujących **wolne bloki pamięci** o różnych długościach oraz mechanizm wymiatania.

[8] **Podział statyczny na stałe partycje**



a. partycje o stałym rozmiarze z osobnymi kolejkami wejściowymi,
b. partycje o stałym rozmiarze z jedną kolejką wejściową.

[9] **Podział dynamiczny**

Do realizacji dynamicznego podziału pamięci można stosować następujące hipotetyczne funkcje:

- **przydziel**(rozmiar, adres)

  - wybranie spośród bloków wolnych bloku pokrywającego zapotrzebowanie,
  - efektem jest zwrócenie adresu wybranego bloku i usunięcie przydzielonego bloku ze zbioru bloków wolnych,
  - **malloc**(size), **calloc**(n, size), **realloc**(ptr, size).

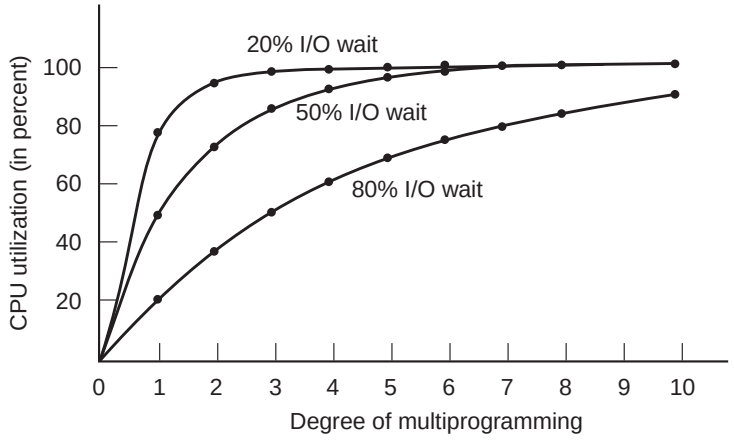
- **zwolnij**(adres) - dołączenie bloku zajętego do zbioru wolnych bloków, np. **free**(ptr),
- **informuj**() - zwróc rozmiar aktualnie największego wolnego bloku, np. **msize**().

[10] **Zarządzanie w kontekście wieloprogramowania**

Dwa zasadnicze aspekty:

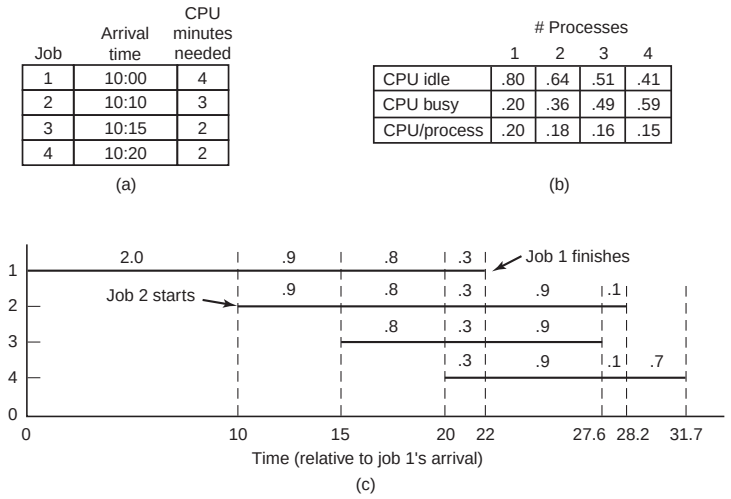
- obsługa relokacji,
  - nie ma pewności gdzie dokładnie program zostanie załadowany do pamięci,
  - wykorzystanie rejestrów bazowych i rejestrów ograniczających.
- wzajemna ochrona pamięci.

[11] **Modelowanie wieloprogramowania**



Wykorzystanie procesora w funkcji liczby procesów w pamięci.

[12] **Analiza wydajności wieloprogramowania**

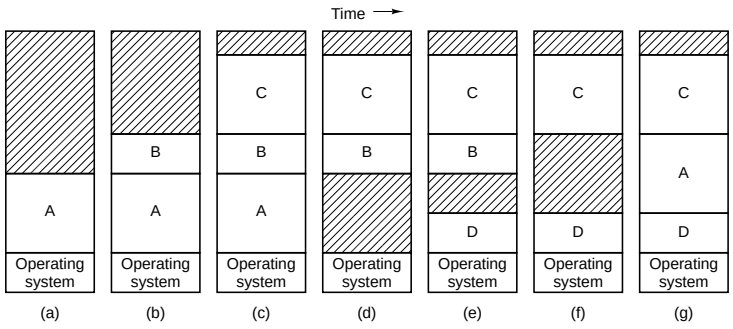


a. czas nadejścia i wymagania poszczególnych prac,

b. wykorzystanie procesora dla procesów z 80% I/O,

c. sekwencja zdarzeń z pokazanym w minutach czasem, jaki dany proces otrzymuje w podanym interwale.

[13] **Wymiatanie (ang. *swapping*)**



Zmiany przydziału pamięci wraz z pojawianiem się i znikaniem procesów z pamięci. Obszary zakreskowane są niewykorzystane.

[14] **Fragmentacja**

Przy wyborze algorytmu alokacji należy uwzględnić następujące czynniki:

- prędkość,

- prostotę,
- efekt fragmentacji.

Fragmentacja wewnętrzna - zjawisko tworzenia niewykorzystywalnych, choć **przydzielonych** w ramach pewnej struktury (partycja, ramka), obszarów pamięci.

Fragmentacja zewnętrzna - zjawisko tworzenia niewykorzystywalnych, **nieprzydzielonych** obszarów pamięci, zazwyczaj spowodowane niedoskonałością działania organizacji alokacji pamięci procesom uzytkowym.

[15] Algorytmy alokacji

Zadaniem algorytmu alokacji jest wybranie wolnego bloku w celu przydzielenia procesowi pamięci.

Wybrane algorytmy alokacji:

- **algorytm pierwszej zgodności** (ang. *First Fit*), wybór pierwszego pasującego ze zbioru/ listy,
- **algorytm najlepszej zgodności** (ang. *Best Fit*), wybór najmniejszego wystarczającego bloku ze zbioru/ listy,
- **algorytm najgorszej zgodności** (ang. *Worst Fit*), wybór największego ze zbioru/ listy,
- **algorytm bliźniaków** (ang. *Buddies*), podział pamięci (o długości 2^k) na dwa równe bliźniacze bloki; połowienie jednego aż do uzyskania bloku o minimalnej długości spełniającego zapotrzebowanie.

[16] Przeciwdziałanie fragmentacji

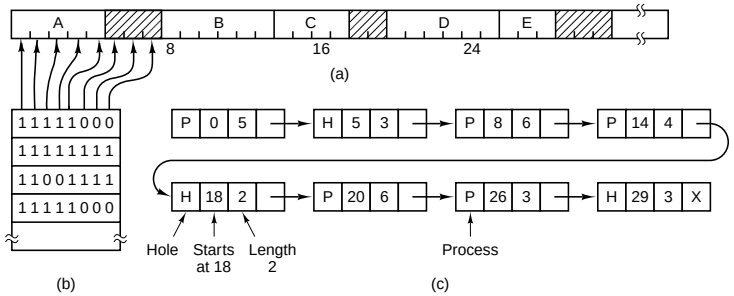
W systemie może być zdefiniowana wartość Δn będąca minimalna dopuszczalną wartością niejawnie przydzielanego bloku, co ułatwia zarządzanie i zwiększa efektywność, ale może prowadzić do zjawiska fragmentacji.

Efektom fragmentacji można przeciwdziałać poprzez:

- zwalnianie i scalanie,
- zagęszczanie i relokacje,

- mechanizm stronicowania.

[17] **Zarządzanie pamięcią - mapy bitowe i listy**

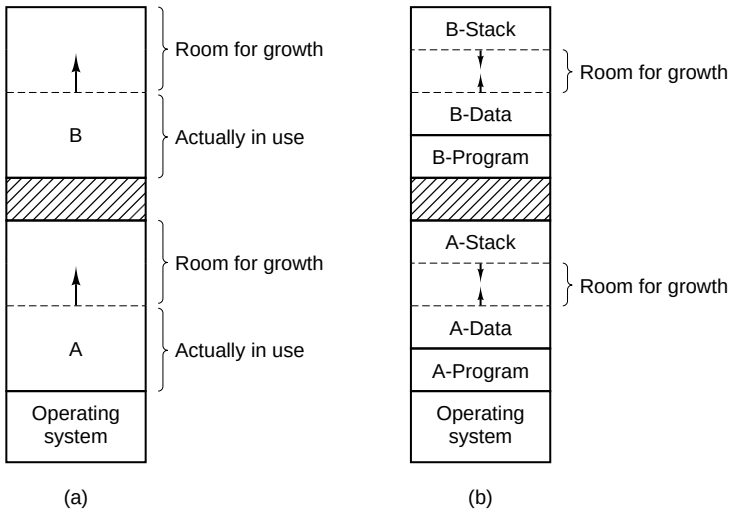


a. fragment pamięci z pięcioma procesami i trzema dziurami,

b. odpowiadająca mapa bitowa,

c. ta sama informacja w postaci listy.

[18] **Problem dynamicznej alokacji**



a. alokacja pamięci dla rosnącego segmentu danych,

b. alokacja pamięci dla rosnącego stosu i rosnącego segmentu danych.

[19] **Pamięć wirtualna**

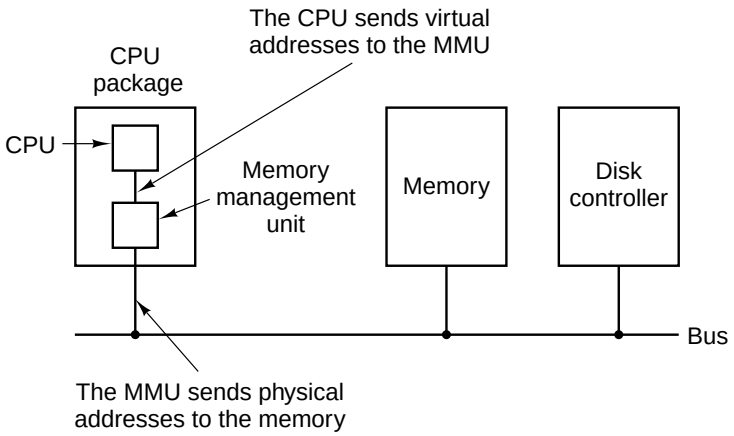
Pamięć wirtualna system pamięci złożony z co najmniej dwóch rodzajów pamięci: *małej i szybkiej* (np. **pamięci operacyjnej**) oraz *dużej, lecz wolnej* (np. **pamięci pomocniczej**), a także z dodatkowego sprzętu i oprogramowania umożliwiającego automatyczne przenoszenie fragmentów pamięci z jednego rodzaju pamięci do drugiego.

Pamięć wirtualna ma być prawie tak szybka jak szybsza z pamięci i prawie tak duża jak większa z pamięci.

Metody realizacji pamięci wirtualnej:

- stronicowanie,
- segmentacja,
- stronicowanie z segmentacją.

[20] **Rola jednostki zarządzającej pamięcią**



MMU (ang. *Memory Management Unit*) może być zintegrowane z procesorem (tak zazwyczaj teraz), ale może być też niezależne (tak zazwyczaj było dawniej).

[21] **Stronicowanie**

Stronicowanie bazuje na stałym podziale pamięci. Jednostki podziału:

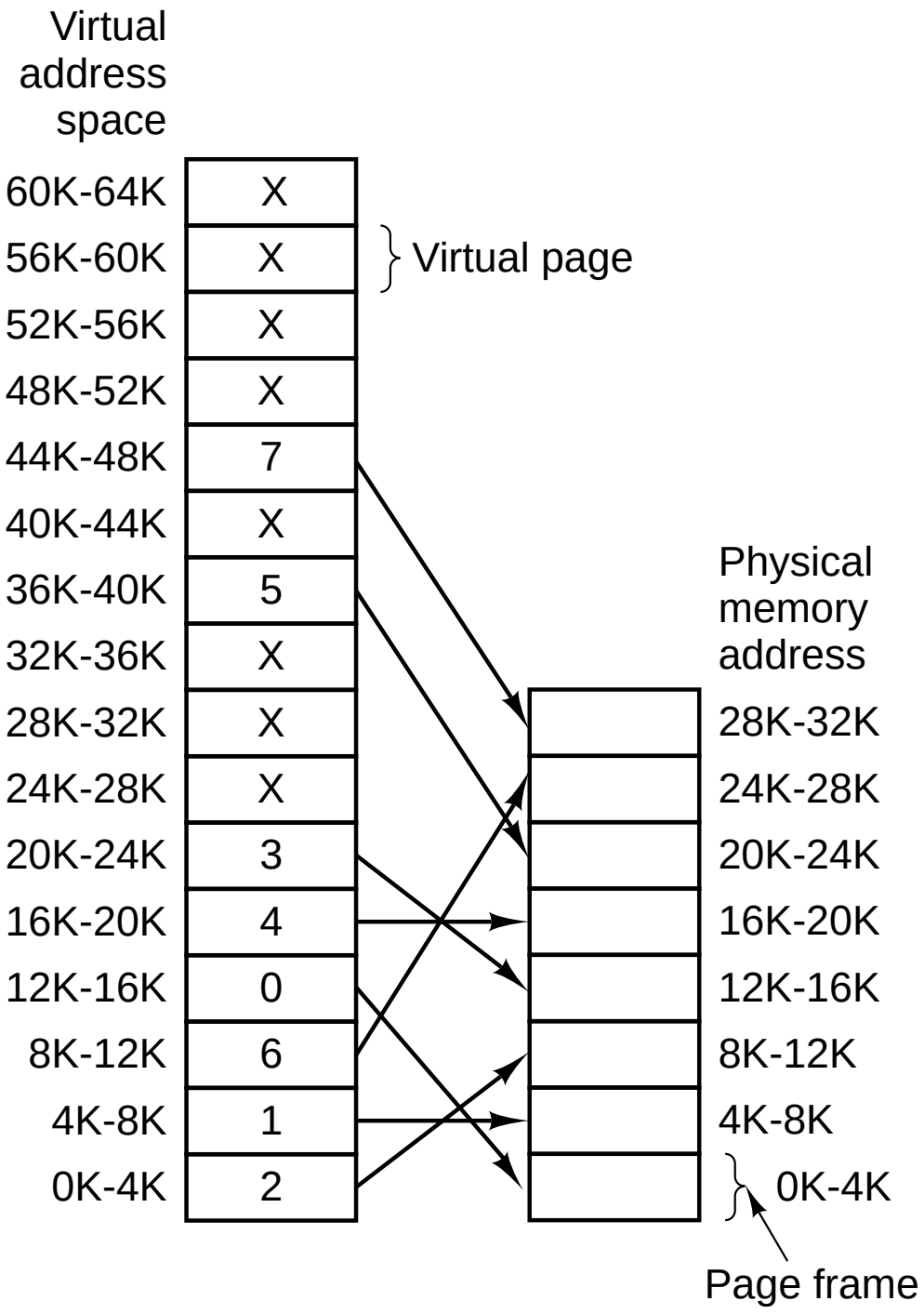
- **ramy, ramki** (ang. *frames*) dla pamięci fizycznej,

- **strony** (ang. *pages*) dla wirtualnej przestrzeni adresowej procesu.

Zadania mechanizmu stronicowania:

- odwzorowywanie adresów wirtualnych w adresy rzeczywiste:
  1. określenie, do której strony odnosi się adres w programie,
  2. znalezienie ramy, którą aktualnie zajmuje dana strona.
- przesyłanie - w zależności od potrzeby - stron z pamięci pomocniczej do pamięci operacyjnej oraz odsyłanie już nie używanych stron z powrotem.

[22] **Przykład translacji**



[23] **Realizacja stronicowania**

Korzystanie z pamięci wirtualnej nie stawia żadnych dodatkowych wymagań użyt-

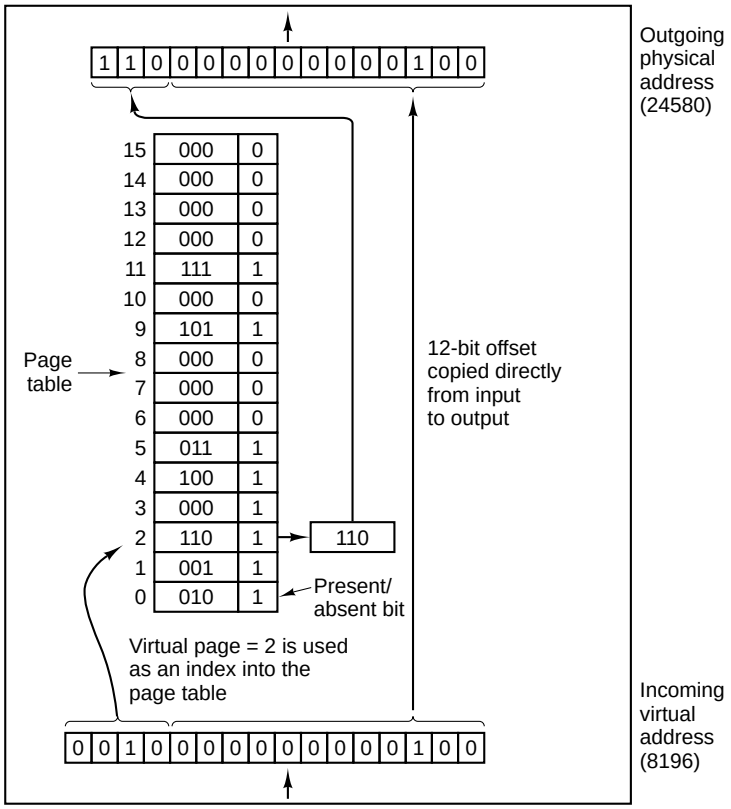
kownikowi. Przydział pamięci realizowany jest systemowo z wykorzystaniem **tablic stron** i/ lub **tablic ram**.

Zapewnienie własności przezroczystości wymaga:

- obsługi przez system przerwania określanego jako **chybienie strony** (ang. *page fault*), sygnalizującego brak w pamięci strony, do której usiłowano się odwołać.
- obsługa chybienia ma zgodnie z przyjętym algorytmem wymiany stron przydzielić ramę i ściągnąć stronę z pamięci pomocniczej,
- w celu minimalizacji strat wydajności odwołań do pamięci wykorzystanie pamięci asocjacyjnych do translacji adresów.

[24] **Przykładowa tablica stron**

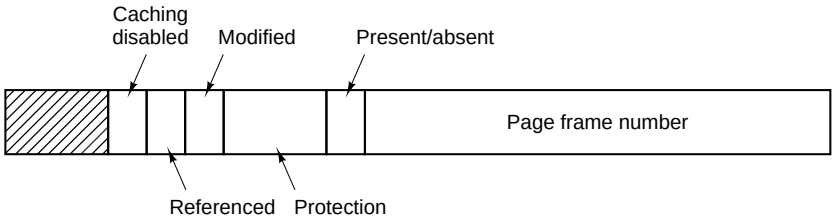
Wewnętrzne operacje MM z szesnastoma stronami po 4kB.



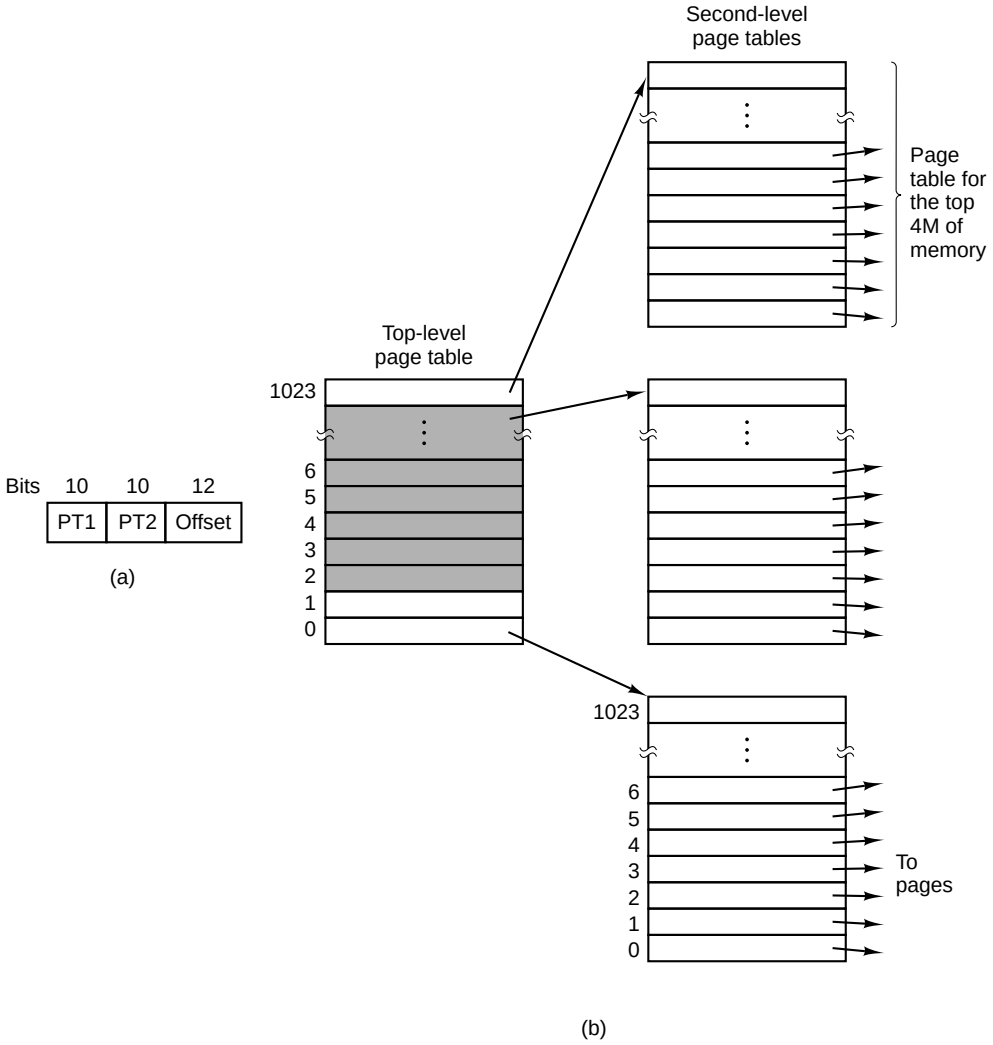
Należy uwzględniać następujące fakty:

- tablica stron może być bardzo duża,
- odwzorowywanie musi być bardzo szybkie.

[25] **Typowy element tablicy stron**



[26] **Wielopoziomowe tablice stron**



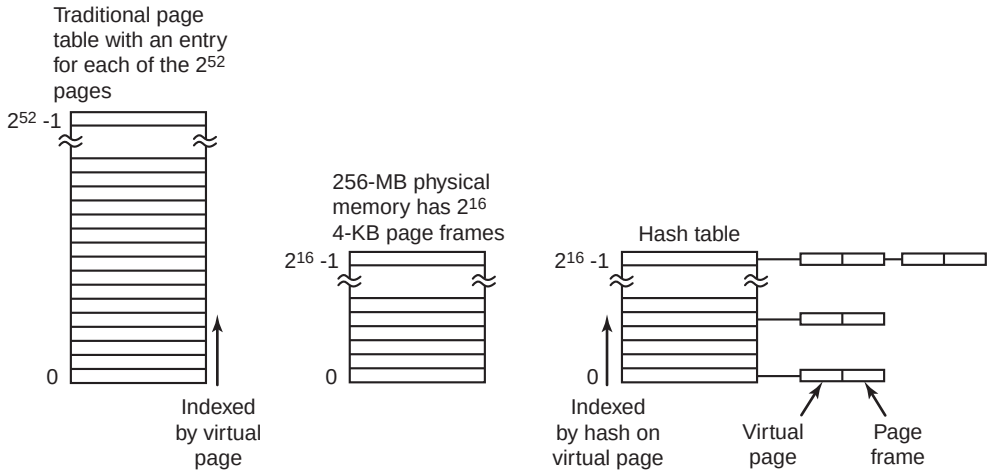
[27] **TLB (ang. *Translation Lookaside Buffer*)**

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Główny cel: minimalizacja liczby odwołań do pamięci w trakcie stronicowania. Aspekty dotyczące TLB:

- koszt przełączania kontekstu,
- koszt wielopoziomowości tablic stron,
- realizacja: sprzętowa/ programowa.

[28] **Odwrócone tablice stron**



[29] **Optymalny rozmiar strony**

Dobór rozmiaru strony ma wpływ na fragmentację wewnętrzną oraz rozmiar tablicy stron.

Niech:

s przeciętny rozmiar procesu w bajtach,

p rozmiar strony,

e rozmiar pozycji w tablicy stron.

wtedy $nadmiar = s * e/p + p/2$

Wyszukiwanie ekstremum, pochodna względem p przyrównana do 0:

$-se/p^2 + 1/2 = 0$

Zatem optymalny rozmiar strony wynosi $p = \sqrt{2se}$

[30] **Obsługa chybienia**

Obsługa chybienia przebiega następująco:

1. wskazanie ramy dla żądanej strony,
2. zwolnienie wskazanej ramy,
3. ściągnięcie żądanej strony w obszar wskazanej ramy,
4. aktualizacja tablicy ram.

Wskazaniem ramy, w przypadku braku ram wolnych, zajmuje się **algorytm wymiany stron** (ang. *page replacement algorithm*). Główne zadanie:

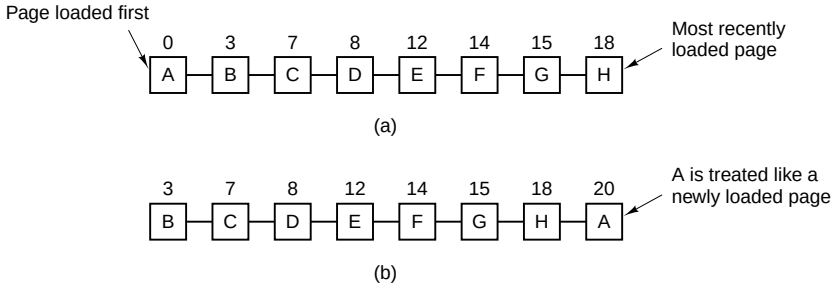
- usunięcie z pamięci operacyjnej strony najmniej potrzebnej w przyszłości, tzn. strony, do której odwołanie nastąpi najpóźniej bądź w ogóle nie nastąpi.

[31] **Algorytmy wymiany stron**

Zestawienie wybranych algorytmów wymiany stron:

- algorytm optymalny,
- algorytm NRU (ang. *not recently used*) (bity R i M),
- algorytm FIFO,
- algorytm drugiej szansy,
- algorytm zegarowy,
- algorytm LRU (ang. *least recently used*).

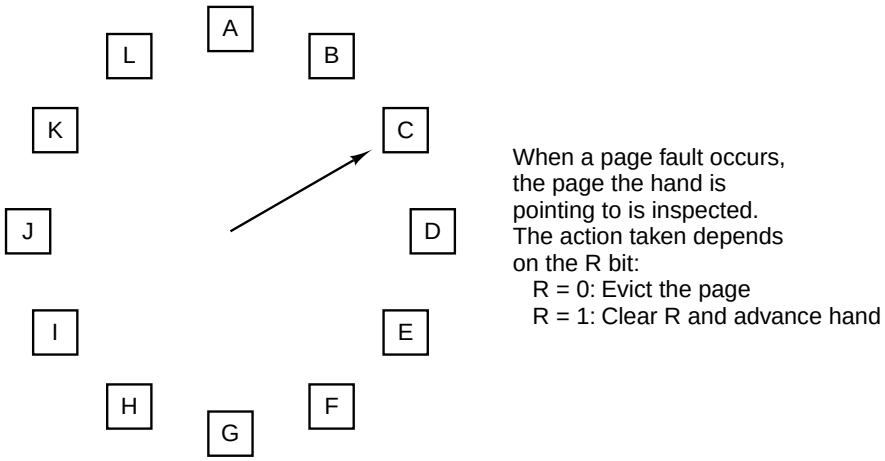
[32] **Algorytm drugiej szansy**



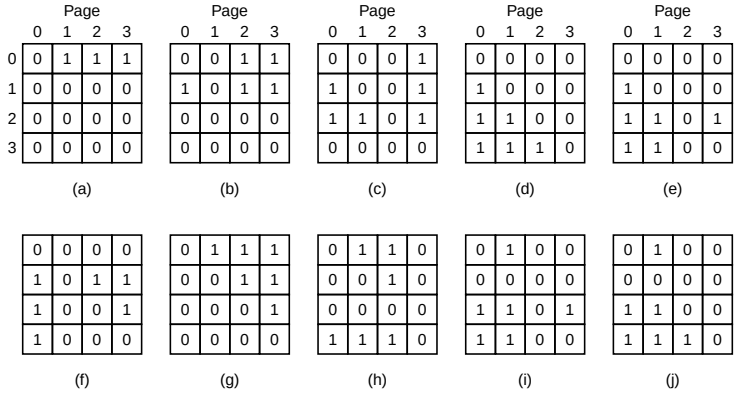
a. strony sortowane w porządku FIFO,

b. nowa postać listy, gdy błąd strony wystąpił w momencie 20 i *A* miał ustawiony bit *R*.

[33] **Algorytm zegarowy wymiany stron**



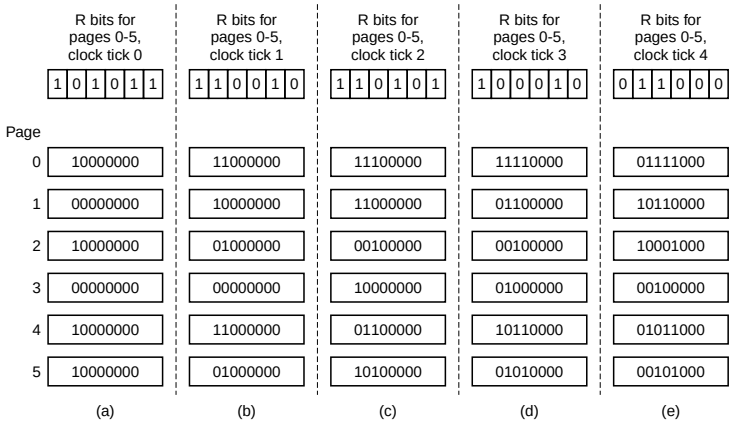
[34] **Sprzętowy algorytm LRU**



Do stron odwoływano się w następującej kolejności:

0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

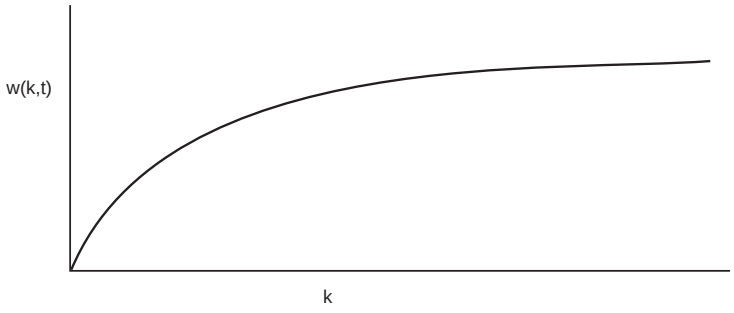
[35] **Programowa symulacja algorytmu LRU (II)**



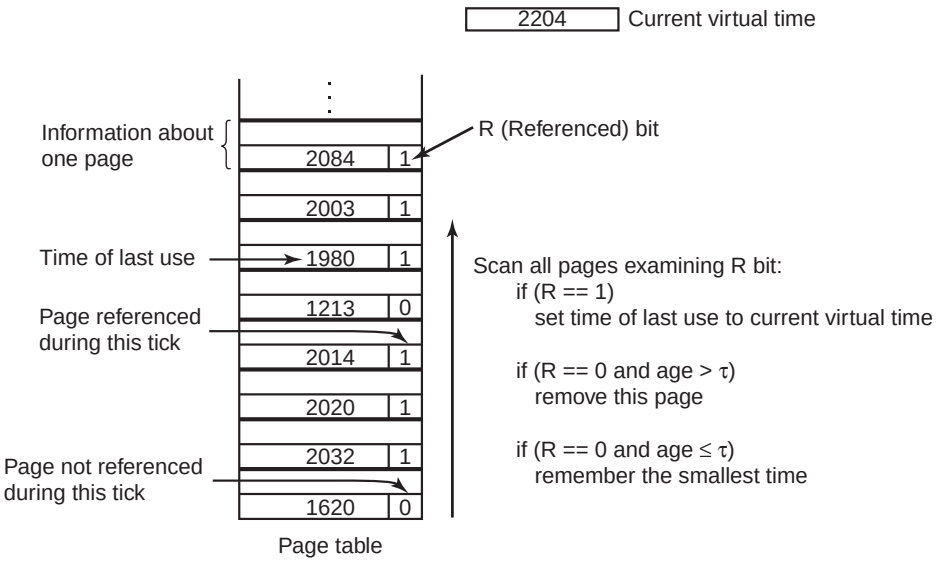
Algorytm postarzania, pokazano sześć stron w okresie pięciu tyknięć zegara.

[36] **Zbiór roboczy stron**

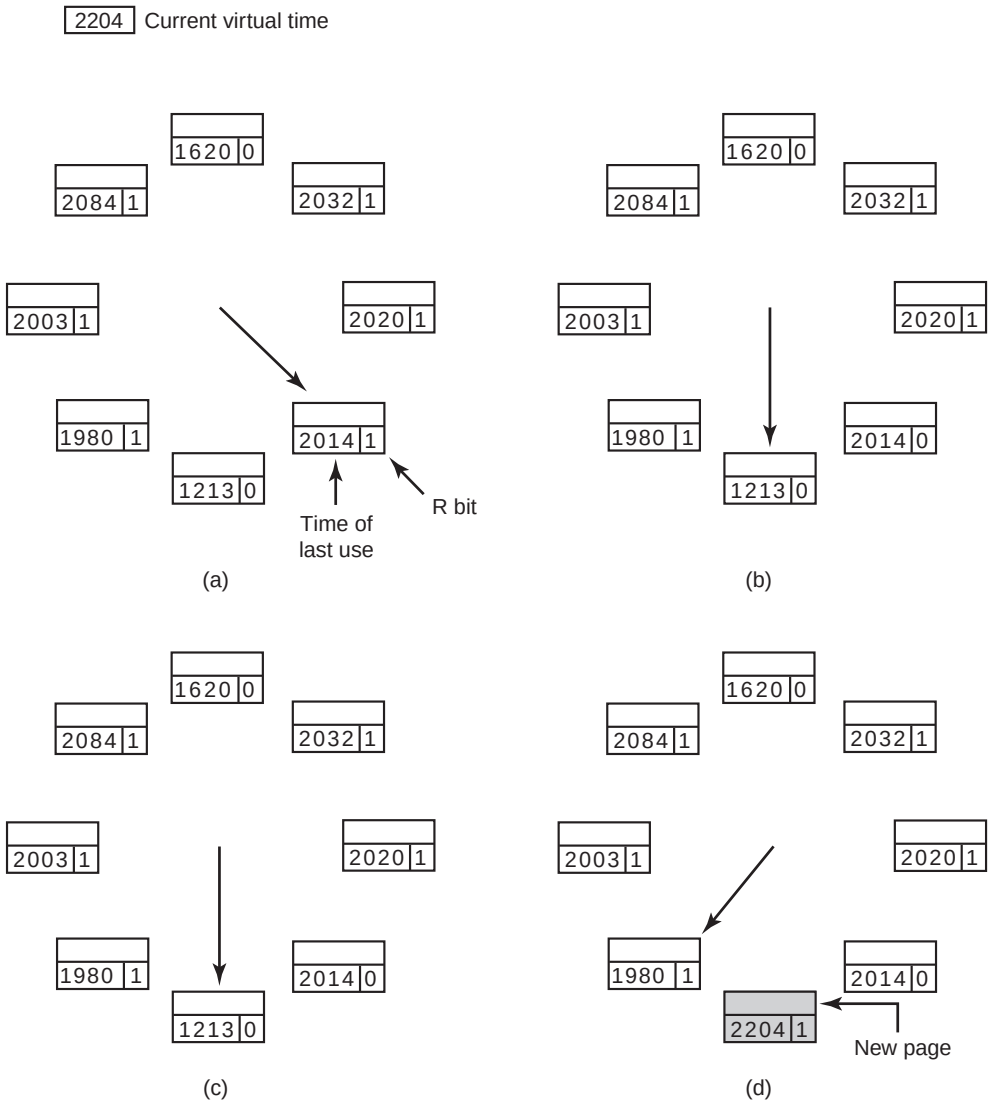
Zbiorem roboczym nazywamy zbiór stron wykorzystywanych w trakcie k ostatnich odwołań do pamięci. Funkcja $w(k,t)$ odpowiada mocy zbioru w chwili t.



[37] **Algorytm zbioru roboczego**



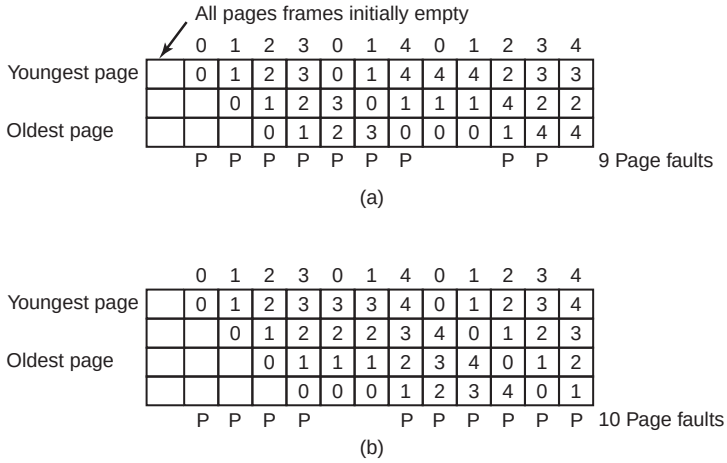
[38] **Zegarowy algorytm zbioru roboczego**



[39] **Porównanie algorytmów wymiany stron**

Algorytm	**Komentarz**
Optymalny	nieimplementowalny, punkt odniesienia
NRU	surowy
FIFO	może wymiatać istotne strony
Drugiej szansy	duże ulepszenie FIFO
Zegarowy	realistyczny
LRU	dobry, ale trudny do implementacji
NFU	dość ogólna aproksymacja LRU
Postarzanie	efektywny z dobrą aproksymacją LRU
Zbiór roboczy	dość kosztowny w implementacji
WSClock	dobry efektywny algorytm

[40] **Anomalia Belady'ego**

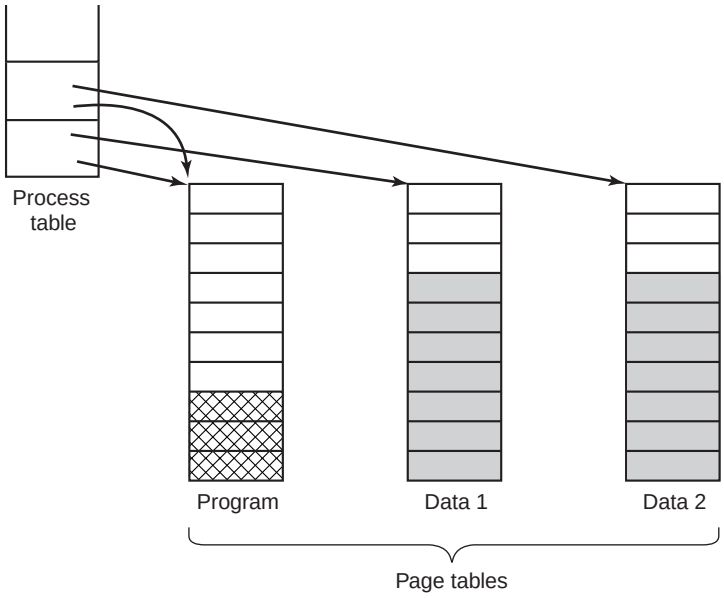


a. algorytm FIFO wymiany stron na maszynie z trzema ramkami,

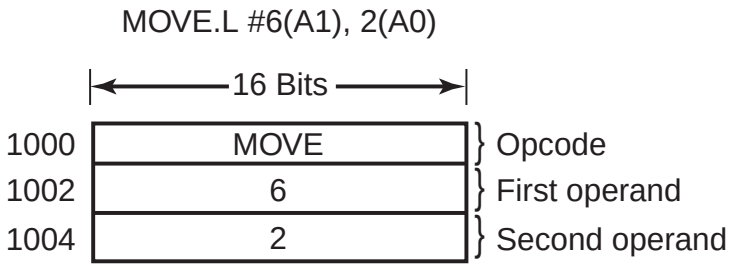
b. algorytm FIFO wymiany stron na maszynie z czteroma ramkami.

- więcej błędów strony przy większej liczbie ramek.

[41] **Strony współdzielone**

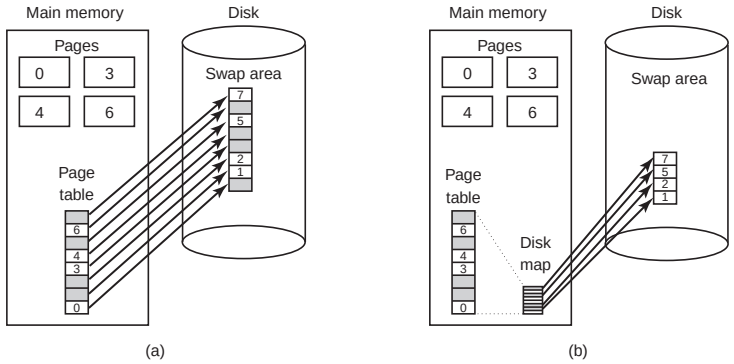


[42] **Obsługa błędu strony**



- instrukcja wywołująca błąd strony,
- odkąd zacząć powtarzanie instrukcji?

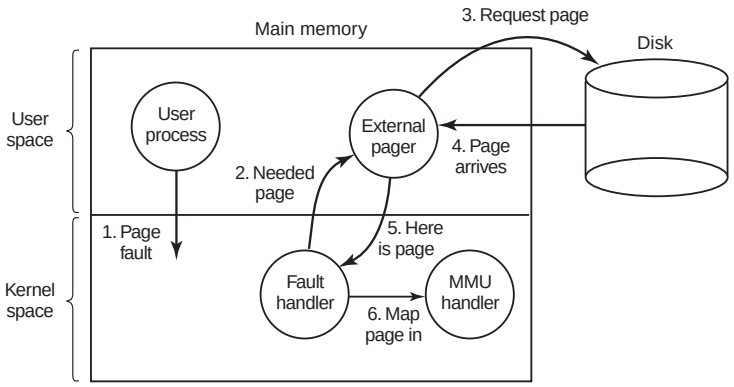
[43] **Organizacja przestrzeni wymiany**



a. stronicowanie do statycznego obszaru wymiany,

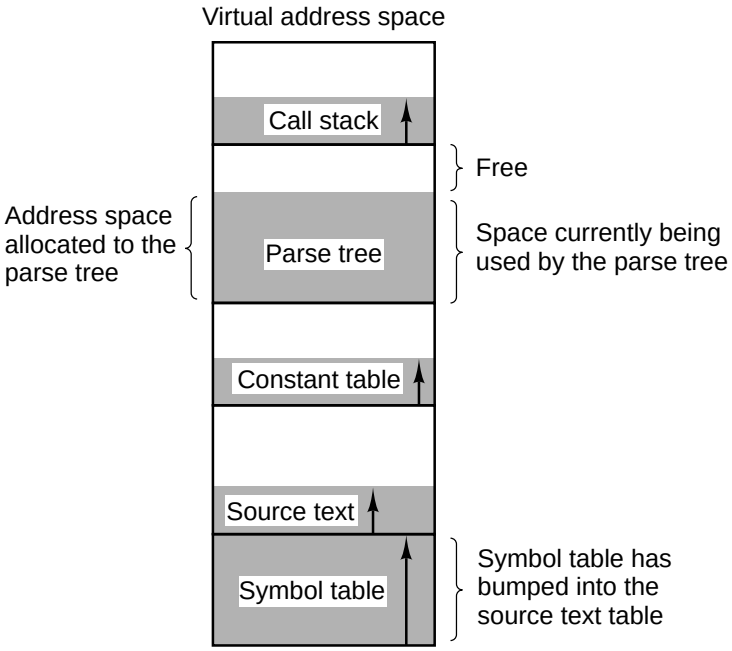
b. dynamiczny zrzut stron.

[44] **Rozdzielenie strategii i mechanizmu realizacji**



- obsługa błędu strony przez zewnętrzny program typu pager.

[45] **Pojedyncza przestrzeń adresowa**



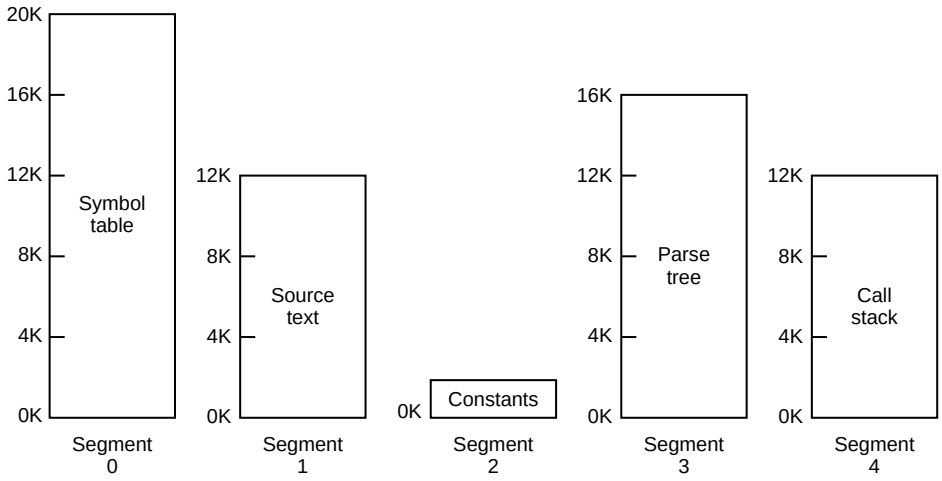
Niewygody pojedynczej przestrzeni adresowej na przykładzie mapy pamięci translatora/ kompilatora.

[46] **Cechy segmentacji**

- celem segmentacji organizacja przestrzeni adresowej w sposób odzwierciedlający logiczny podział informacji,
- przy organizacji przestrzeni wirtualnej możliwość korzystania z dużej liczby segmentów z nadawanymi przez programistę nazwami,
- przestrzeń adresowa dwuwymiarowa z racji identyfikacji adresu poprzez parę *nazwa segmentu + adres wewnątrz segmentu*,
- odzworowanie adresu organizowane zazwyczaj poprzez osobną dla każdego procesu **tablicę segmentów**,
- elementy tablicy segmentów określane mianem **deskryptorów segmentu**,

- każdy deskryptor zawiera **adres bazy** oraz **rozmiar** segmentu.

[47] **Rozmiary segmentów**



Segmentacja pamięci umożliwia każdej tablicy niezależną zmianę rozmiaru.

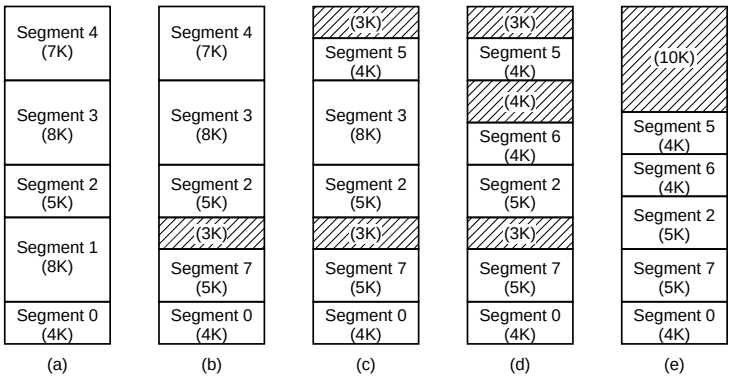
[48] **Stronicowanie a segmentacja (I)**

- celem segmentacji **logiczny** podział pamięci operacyjnej, celem stronicowania **fizyczny** podział pamięci,
- stronicowanie mechanizmem niskiego poziomu, niewidocznym dla programisty; segmentacja mechanizmem wyższego poziomu, widocznym dla programisty,
- rozmiar strony stały, ustalona, wynikający z architektury, rozmiar segmentu dowolny określany przez programistę,
- zarówno w przypadku zastosowania stronicowania jak i segmentacji całkowita przestrzeń adresowa może być większa od dostępnej pamięci fizycznej,
- segmentacja umożliwia lepszą ochronę poszczególnych elementów procesu poprzez możliwość rozróżniania segmentów logicznie grupujących elementy procesów,

[49] **Stronicowanie a segmentacja (II)**

- segmentacja umożliwia łatwiejsze zarządzanie elementami procesu o zmiennym rozmiarze (np. stos, sterta),
- segmentacja udostępnia współdzielenie procedur między procesami (np. segmenty dzielone, a przez to biblioteki dzielone).
- główny cel wprowadzenia stronicowania: uzyskanie większej pamięci adresowej bez konieczności zakupu dodatkowej pamięci fizycznej,
- główny cel wprowadzenia segmentacji: umożliwienie rozdzielenia programów i danych na logicznie niezależne przestrzenie adresowe z ułatwieniem współdzielenia wybranych obszarów i lepszej ochrony.
- segmentacja - może zachodzić fragmentacja zewnętrzna,
- stronicowanie - może zachodzić fragmentacja wewnętrzna.

[50] **Implementacja czystej segmentacji**



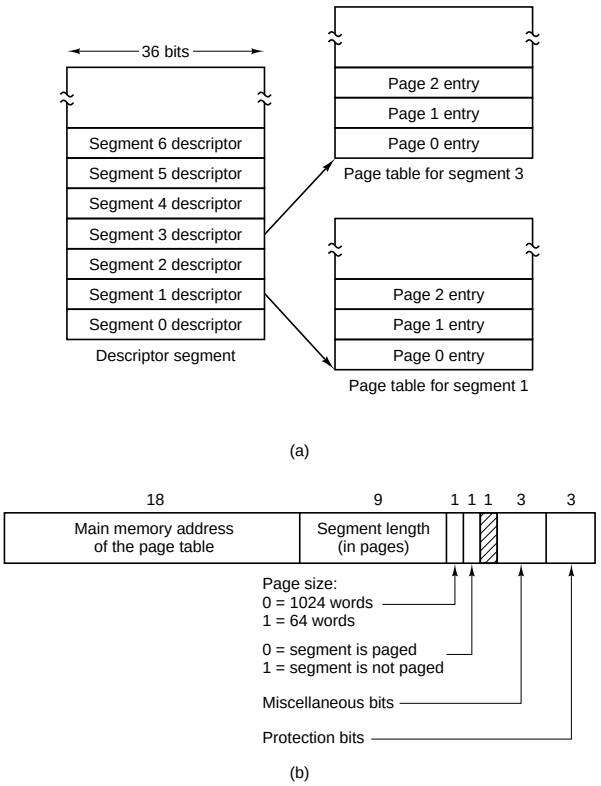
- zjawisko **fragmentacji zewnętrznej** (ang. *external fragmentation, checkerboarding*),
- zbijanie/ gromadzenie segmentów metodą przeciwdziałania fragmentacji (rys. e),
- wykorzystanie stronicowania metodą przeciwdziałania fragmentacji.

[51] **MULTICS: Segmentacja ze stronicowaniem**

- komputery Honeywell 6000 i następne,

- maksymalnie możliwych 2^{18} segmentów, z których każdy mógł mieć maksymalnie 65536 36-bitowych słów,
- stronicowane segmenty, dla każdego procesu tablica segmentów, sama tablica segmentów również stronicowanym segmentem,
- fizyczne adresy 24-bitowe, strony rozmieszczane z dopasowaniem do 64 (2^6) bajtów, zatem 18 bitów na adres tablicy stron,
- standardowy rozmiar stron 2^{10} słów, część wewnętrznych małych segmentów systemu niestronicowana bądź ze stronami po 64 słowa,
- ewentualny adres segmentu w pamięci pomocniczej w osobnej tablicy wykorzystywanej przez program obsługi chybienia strony.

[52] **MULTICS: Pamięć wirtualna**

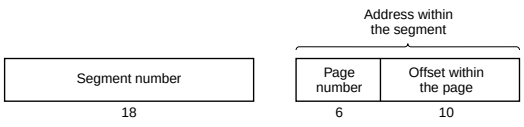


Pamięć wirtualna systemu Multics

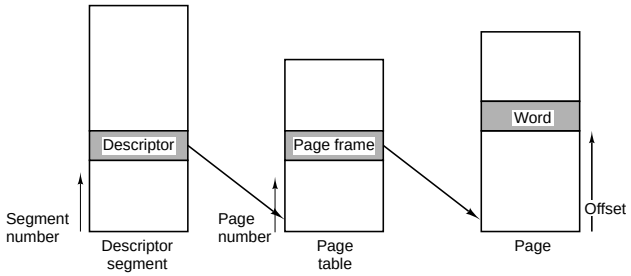
a. segment deskryptorów ze wskazaniami do tablic stron,

b. deskryptor segmentu.

[53] **MULTICS: Wyznaczanie adresu fizycznego**



34-bitowy adres wirtualny w systemie Multics.



Translacja dwuczęściowego adresu wirtualnego na adres w pamięci głównej. Dla przejrzystości pominięto fakt, że segment deskryptorów jest również stronicowany.

[54] **MULTICS: bufory TLB**

Comparison field					Is this entry used?
Segment number	Virtual page	Page frame	Protection	Age	
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

Uproszczona wersja buforów TLB. 16 pozycji, dla których numer segmentu i strony są porównywane równolegle. W rzeczywistości ze względu na zróżnicowanie rozmiaru stron struktura jest bardziej złożona.

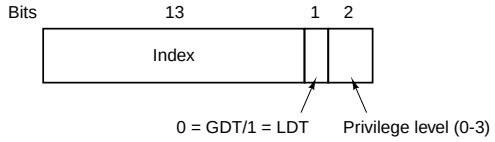
[55] **Intel Pentium: Segmentacja ze stronicowaniem**

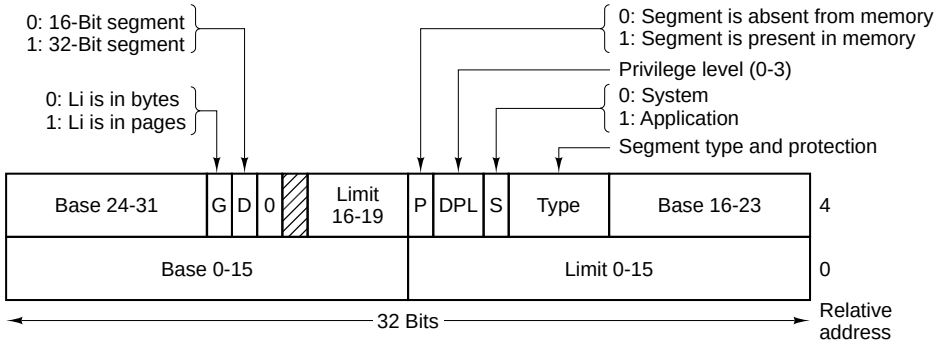
- Multics: 256K niezależnych segmentów, każdy do 64K 36-bitowych słów,
- Pentium: 16K niezależnych segmentów, każdy do 1G 32-bitowych słów,

- pojedyncza **GDT** (ang. *Global Descriptor Table*) w systemie,
- **LDT** (ang. *Local Descriptor Table*) dla każdego procesu.

[56] **Intel Pentium: Selektor i deskryptor segmentu**

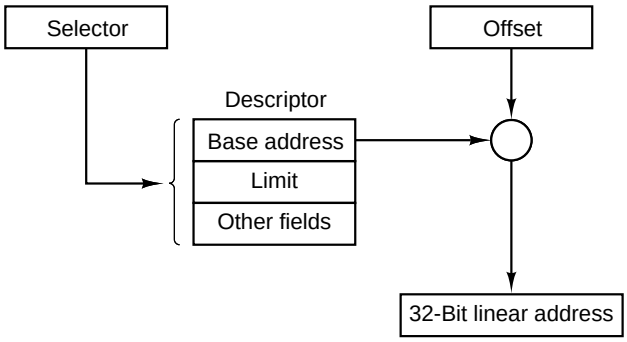
Selektor w Pentium





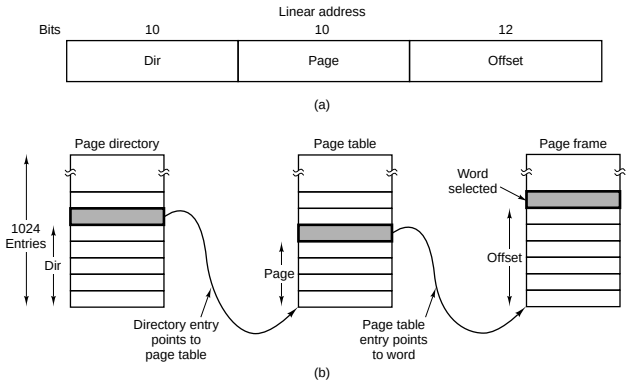
Deskryptor segmentu kodu w Pentium

[57] **Intel Pentium: Wyznaczanie adresu fizycznego (I)**



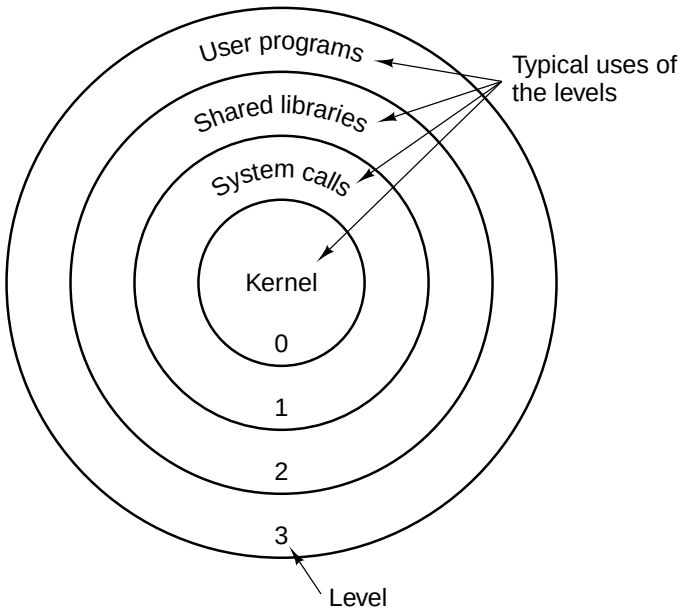
- jeżeli stronicowanie wyłączone (czysta segmentacja), uzyskany adres to adres fizyczny,
- przy włączonym stronicowaniu interpretacja uzyskanego adresu jako wirtualnego,
- wykorzystanie buforów TLB,

[58] **Intel Pentium: Wyznaczanie adresu fizycznego (II)**



- 32-bitowa logiczna przestrzeń adresowa, strony o rozmiarze 4KB,
- każdy proces posiada katalog stron z 2^{10} pozycjami, z których każda wskazuje na tablicę stron również z 2^{10} pozycjami.
- możliwa pojedyncza, stronicowana, 32-bitowa przestrzeń adresowa na potrzeby aplikacji.

[59] **Intel Pentium: Ochrona**



- cztery poziomy ochrony, można odwoływać się do danych segmentów swojego i wyższych poziomów,

- aby odwołać się do procedury innego niż własny poziomu CALL ma jako argument nie adres ale selektor do tzw. *call gate*.