Zarządzanie pamięcią

Ostatnia modyfikacja: 15.04.2020

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

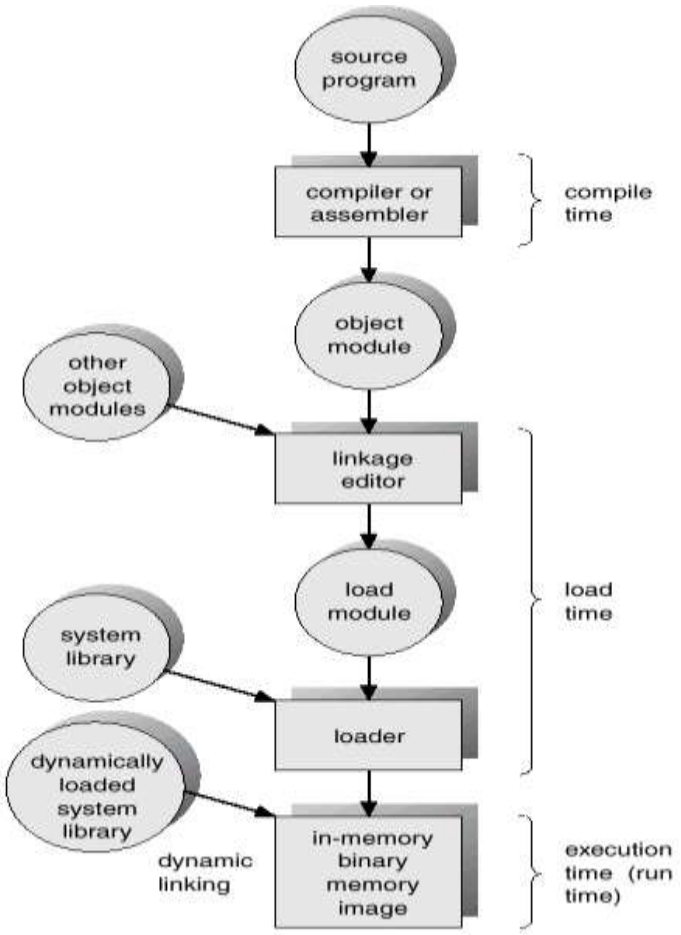
Spis treści

Wprowadzenie

- Procesor może bezpośrednio sięgać jedynie do pamięci operacyjnej i pamięci podręcznej (*cache*)
- Program musi zostać przeniesiony do pamięci operacyjnej aby mógł być wykonany (jako proces).
- *Kolejka wejściowa procesów* – zbiór procesów oczekujących na dysku na przeniesienie do pamięci operacyjnej w celu wykonania.
- Do jednostki pamięci dociera tylko strumień adresów, a nie informacja w jaki sposób te adresy zostały wytworzone.
- Pojedynczy dostęp do pamięci może trwać przez okres wielu cykli procesora (*stall*)
- Pamięć podręczna (*cache*) jest buforem przesłań pomiędzy rejestrami procesora a pamięcią główną
- Programy użytkownika przechodzą kilka faz (kompilacja, konsolidacja, ładowanie, łączenie dynamiczne) zanim są wykonane.

Wiązanie adresów logicznych z adresami pamięci operacyjnej

- **Kompilacja**: Jeśli podczas kompilacji znane jest miejsce, w którym proces będzie przebywał w pamięci, to można wygenerować *kod bezwzględny*. Domyślnie adresy są *względne* dla każdego modułu kompilacji.
- **Konsolidacja**.
  - *statyczna* - odniesienia do innych modułów zamienione są na adresy przed załadowaniem programu
  - *dynamiczna w czasie ładowania* – w czasie ładowania programu następuje znalezienie (po ew. załadowaniu) modułów bibliotecznych i związanie odpowiednich adresów
  - *dynamiczna w czasie wykonania* – znalezienie/załadowanie modułów i związanie adresów następuje dopiero przy odwołaniu się do nich w czasie wykonania programu *(dynamic loading).*
- **Ładowanie**: *absolutne* – jeżeli po konsolidacji znane jest miejsce, w którym proces będzie przebywał w pamięci; *relokowalne* – w przeciwnym przypadku (*kod przemieszczalny*).
- **Wykonania**: System może dopuszczać przemieszczanie procesu w pamięci podczas wykonywania (ładowanie dynamiczne w czasie wykonania, *dynamic memory mapping*). Systemy umożliwiają też ładowanie modułów do pamięci na żądanie procesu *(dynamic loading);* proces uzyskuje adresy wejść do tych modułów.



Logiczna i fizyczna przestrzeń adresowa

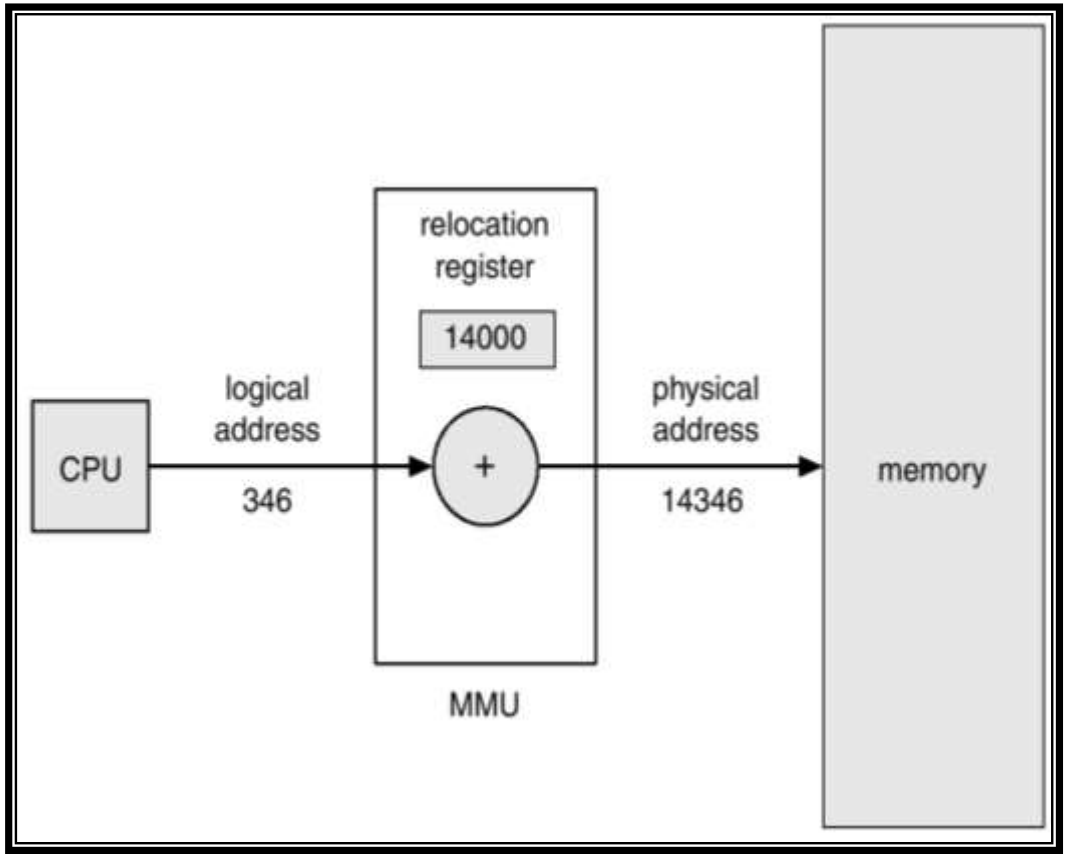
- Adres wytworzony przez proces to adres logiczny; jednostka pamięci używa adresów fizycznych.

- Adresy logiczne są tożsame z fizycznymi, gdy są tworzone w czasie kompilacji, linkowania czy ładowania. Adresy logiczne (wirtualne) i fizyczne różnią się przy wykorzystywaniu wiązania adresów w czasie wykonania.

- Zbiór wszystkich adresów generowanych przez program, to logiczna przestrzeń adresowa. Zbiór adresów fizycznych odpowiadających logicznej przestrzeni adresowej, to fizyczna przestrzeń adresowa.

Jednostka zarządzania pamięcią (MMU)

- MMU, to urządzenie (sprzęt) odwzorowujące adresy wirtualne w fizyczne i chroniące przed niepowołanym dostępem.
- Wartość rejestru przemieszczenia (*relocation register*) jest sumowana z każdym adresem (wirtualnym) generowanym przez proces użytkownika w chwili odwoływania się pamięci.
- Program użytkownika zawsze ma do czynienia z adresami logicznymi; nigdy nie widzi rzeczywistych adresów fizycznych pamięci.



Ładowanie dynamiczne

Zalety:

- Podprogram nie jest wprowadzony do pamięci dopóty, dopóki nie zostanie wywołany (przez inny podprogram).
- Lepsze wykorzystanie pamięci, nieużywane podprogramy nigdy nie są ładowane.
- Przydatne, gdy okazjonalnie trzeba wykonać wielkie fragmenty kodu.
- Nie potrzeba specjalnego wsparcia ze strony systemu operacyjnego (ładowanie zleca program użytkownika).

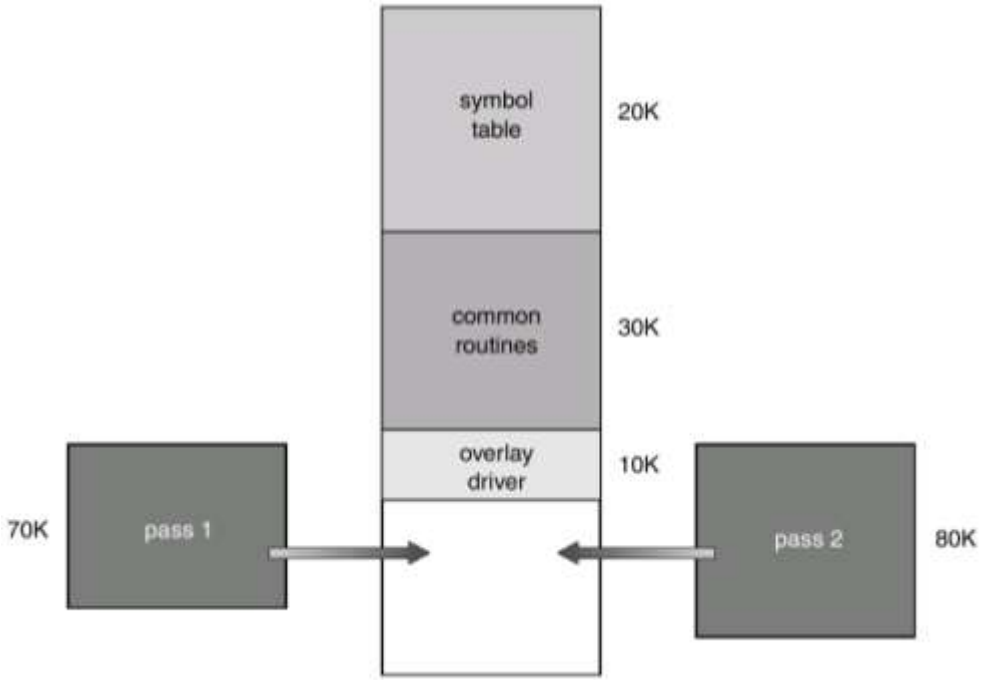
Konsolidacja dynamiczna

- Konsolidacja programu jest opóźniona aż do jego wykonania.
- W obrazie binarnym są namiastki (*stubs)* procedury, używana do lokalizacji procedury w pamięci, bądź załadowania odpowiedniej biblioteki.
- Wykonanie namiastki powoduje sprawdzenie, czy potrzebny podprogram jest już w pamięci; jeśli podprogramu nie ma, to zostanie sprowadzony do pamięci.
- System operacyjny musi wspierać konsolidację dynamiczną, gdyż potrzebny jest dostęp (chroniony) do tych samych obszarów pamięci (instrukcji i danych procedur) przez różne procesy.

Nakładkowanie

- W pamięci są tylko te instrukcje i dane programu, które są potrzebne w danym czasie.
- Potrzebne wówczas, gdy proces wymaga więcej pamięci niż jest fizycznie dostępnej.
- Realizacja przez użytkownika bez specjalnego wsparcia ze strony systemu operacyjnego. Programowanie struktury nakładkowania bywa złożone.
- Przykład: realizacja

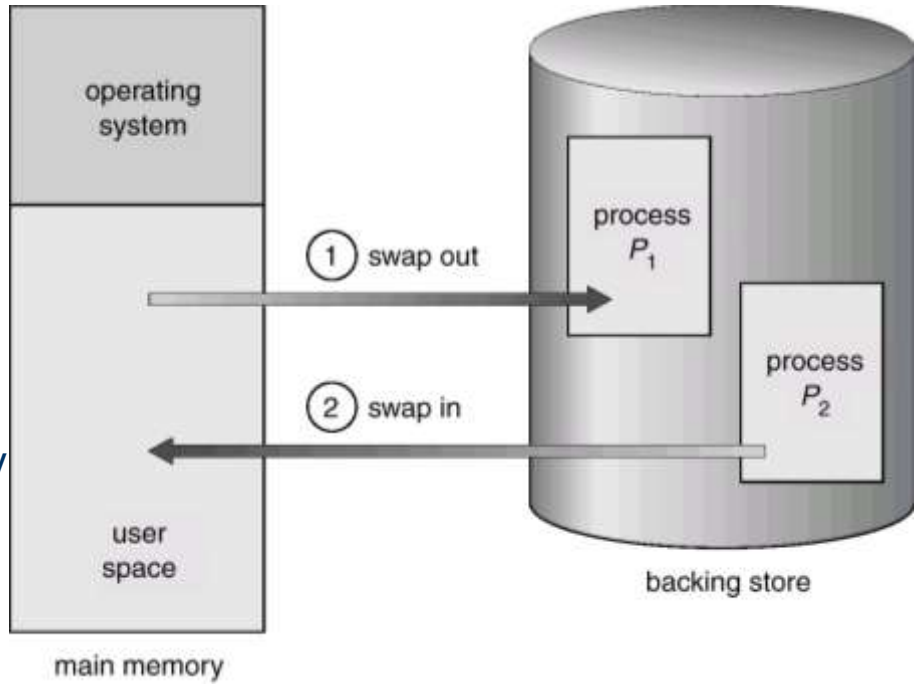
  dwuprzebiegowego kompilatora



Wymiana (*swapping*)

- Proces może być czasowo usuwany z pamięci operacyjnej do pamięci pomocniczej i z powrotem - dla kontynuacji wykonania.

- Urządzenie wymiany - szybki dysk o pojemności na tyle dużej, by pomieścić kopie obrazów pamięci wszystkich procesów w systemie; potrzebny jest dostęp bezpośredni do tych obrazów pamięci.
- *Wytaczanie (roll out) i wtaczanie (roll in) procesu* – wariant polityki wymiany, w którym proces o niskim priorytecie jest usuwany z pamięci na czas wykonania programu o wysokim priorytecie.
- Znaczną część czasu wymiany zajmuje czas przesyłania, bezpośrednio związany z wielkością pamięci podlegającej wymianie.
- Warianty wymiany są zrealizowane w wielu systemach, np. UNIX czy Microsoft Windows.



Wymiana – c.d.

- Czy proces przeniesiony na urządzenie wymiany zostanie przywrócony w to samo miejsce pamięci?
  - To zależy od metody wiązania adresów
  - Trzeba uwzględnić zaległe transfery: bufory procesu – urządzenia wejścia-wyjścia; nie usuwać buforów procesu i przywracać w to samo miejsce pamięci, albo stosować buforowanie w jądrze *(double buffering).*
- Przykładowe koszty wymiany procesu o rozmiarze 100MB przy szybkości przesyłu 50MB/sec
  - Wytaczanie: 2 s, wtaczanie: 2s
  - Narzut na dwukrotne przełączenie kontekstu: 4 s
- Wymiana jest używana we współczesnych systemach operacyjnych w postaci zmodyfikowanej (Unix, Linux, Windows etc.):
  - Domyślnie wymiana jest wyłączona
  - Wymiana jest włączana tylko przy niskim poziomie wolnej pamięci
  - Wymiana jest włączana przy odpowiednio dużym zamówieniu na pamięć przez proces

Wymiana w urządzeniach mobilnych

- Typowo wymiana nie jest realizowana
  - Pamięć *flash*
    - Mała pojemność
    - Ograniczona liczba cykli zapisu
    - Mała przepustowość pamięć *flash*-CPU na platformach mobilnych

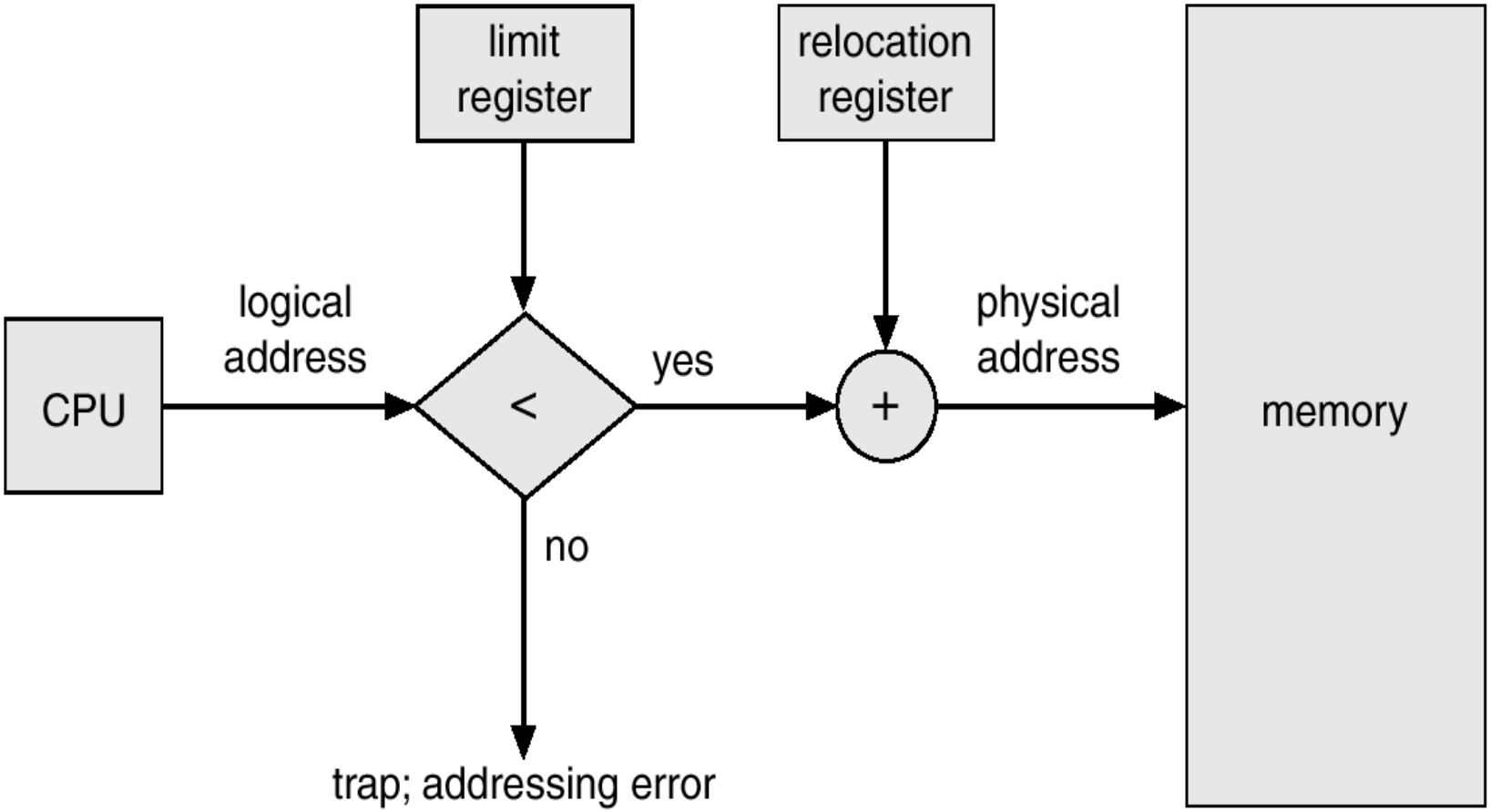
- Metody alternatywne przy małej ilości wolnej pamięci:
  - iOS prosi aplikacje, by dobrowolnie zmniejszyły zużycie pamięci
    - Dane read-only są usuwane, gdyż mogą być ponownie odczytane z pamięci flash – w razie potrzeby
    - Niepowodzenie może skutkować zakończeniem zadania
  - Android kończy aplikację w przypadku niskiego poziomu wolnej pamięci, zapisując wcześniej stan aplikacji – umożliwiając szybkie wznowienie
  - Obydwa systemy wspierają stronicowanie

Ciągły przydział pamięci operacyjnej

- Pamięć operacyjna może być dzielona na dwie części:
  - pamięć rezydującego systemu operacyjnego (zwykle w dolnej części przylegającej do wektora przerwań).
  - pamięć dla procesów użytkownika; może być dzielona na części statycznie lub dynamicznie
  - każdy proces umieszczony jest w spójnym bloku adresów pamięci fizycznej

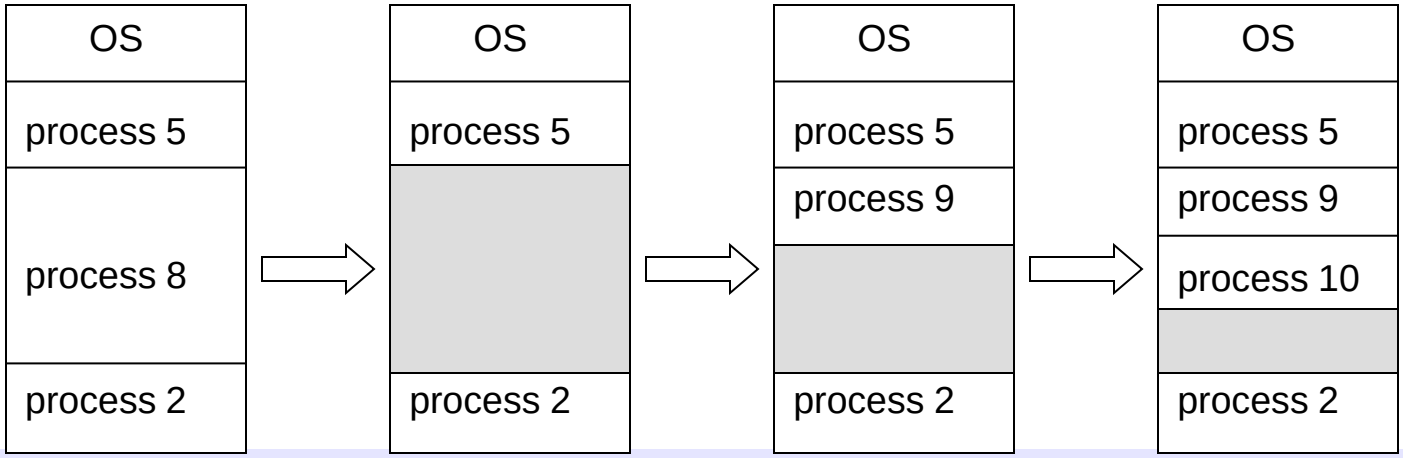
- Pamięć procesu może zawierać spójny zakres adresów
  - ochronę pamięci jednego procesu przed dostępem z innego procesu i ochronę jądra można uzyskać przy użyciu rejestrów sprzętowych i arytmometru adresów.
  - Rejestr przemieszczenia zawiera wartość najmniejszego adresu fizycznego, a rejestr graniczny - górny zakres adresów logicznych.

Ciągły przydział pamięci operacyjnej (c.d.)



Ciągły przydział pamięci operacyjnej (c.d.)

- Przydział wielu obszarów procesowi
  - *Dziura* – nieprzydzielony blok pamięci fizycznej.
  - Kiedy proces ma być załadowany do pamięci - system przydziela mu odpowiednio duży blok pamięci wolnej (dziury).
  - Kiedy proces się kończy – zwalnia swoją partycję. Przyległe wolne partycje mogą być scalane
  - System operacyjne utrzymuje bazę danych o:
    a) obszarach zajętych, b) obszarach wolnych (dziurach)



Dynamiczny przydział pamięci

Jak zrealizować zamówienie na pamięć wielkości *n* za pomocą listy dziur?

- **Pierwsze dopasowanie**: przydzielona jest pierwsza z dziur, która ma odpowiednio duży rozmiar.
- **Najlepsze dopasowanie**: przydzielona jest *najmniejsza* dziura, która ma wystarczająco duży rozmiar. Potrzeba przeglądać całą listę dziur, chyba że są uporządkowane wg rozmiaru. Metoda daje najmniejszą fragmentację zewnętrzną.
- **Najgorsze dopasowanie**: przydzielona jest *największa* dziura, która ma wystarczająco duży rozmiar. Potrzeba przeglądać całą listę dziur, chyba że są uporządkowane wg rozmiaru. Metoda daje największą fragmentację zewnętrzną

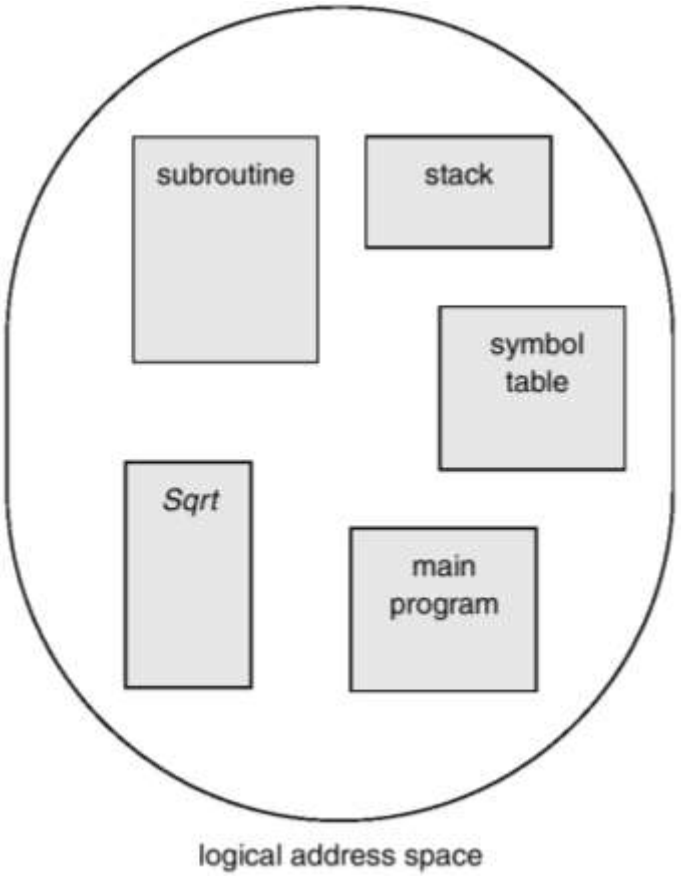
Pierwsze dopasowanie i najlepsze dopasowanie są lepsze niż najgorsze dopasowanie, jeżeli chodzi o szybkość i wykorzystanie pamięci.

Fragmentacja

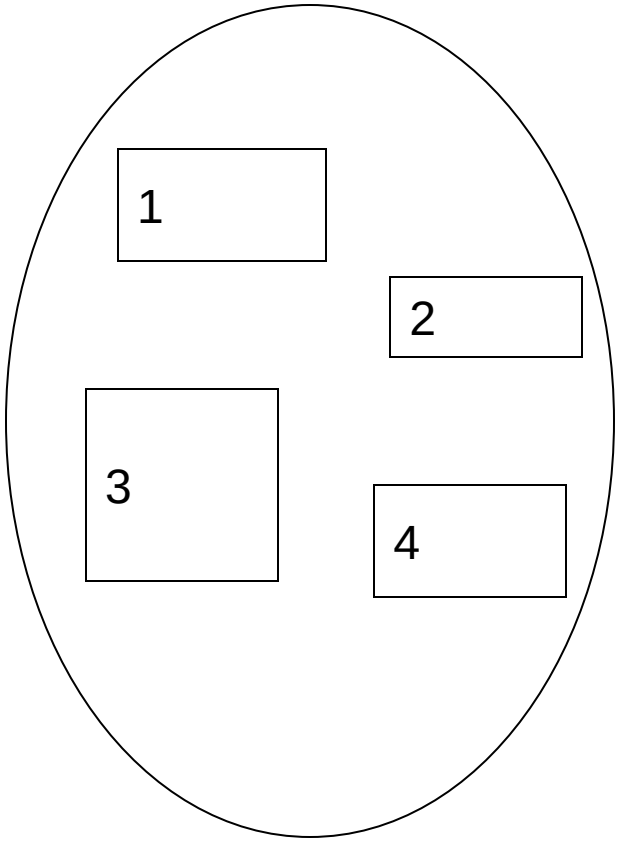
- **Fragmentacja zewnętrzna** - suma pamięci wolnej wystarcza do realizacji zamówienia, ale pamięć wolna nie jest ciągła.
- **Fragmentacja wewnętrzna** – pamięć jest przydzielona w kwantach, które są większe od wielkości żądanej przez proces.
- Analiza statystyczna pokazuje, że dla metody pierwszego dopasowania i przydziału N bloków, następnych *N/2* bloków będzie nieużytecznych ze względu na fragmentację zewnętrzną.
- Fragmentację zewnętrzną można eliminować przez upakowywanie
  - Zapełnione bloki pamięci procesu są przesuwane tak, by tworzyły ciągły obszar alokacji.
  - Upakowanie jest możliwe jedynie wówczas, gdy przesuwanie bloków pamięci jest dynamiczne (w czasie wykonywania procesu).
  - Rozwiązywanie problemów z wejściem/wyjściem:
    - unieruchomić pamięć zadania wykonującego wej/wyj.
    - Używać operacji wej/wyj tylko z buforami systemowymi

Segmentacja

- Segmentacja - schemat zarządzania pamięcią, który urzeczywistnia sposób widzenia pamięci przez użytkownika.
- Program jest zbiorem segmentów, np.:
  program główny,
  procedura,
  funkcja,
  zmienne lokalne, globalne,
  obszar danych wspólnych,
  stos,
  tablica symboli,
  .....



Widoki segmentacji



Widok użytkownika

Przydział segmentów pamięci fizycznej

Architektura segmentacji

- Adres logiczny to para:

  <numer segmentu, przesunięcie>,

- *Tablica segmentów* – zawiera wpisy o następującej zawartości:
  - *baza* – adres fizyczny początku segmentu w pamięci.
  - *granica* – określa długość segmentu.
- *Rejestr bazowy tablicy segmentów (STBR)* wskazuje położenie tablicy segmentów w pamięci.
- *Rejestr długości tablicy segmentów (STLR)* zawiera liczbę segmentów w programie;

  numer *s* segmentu jest poprawny, jeśli *s* < STLR.

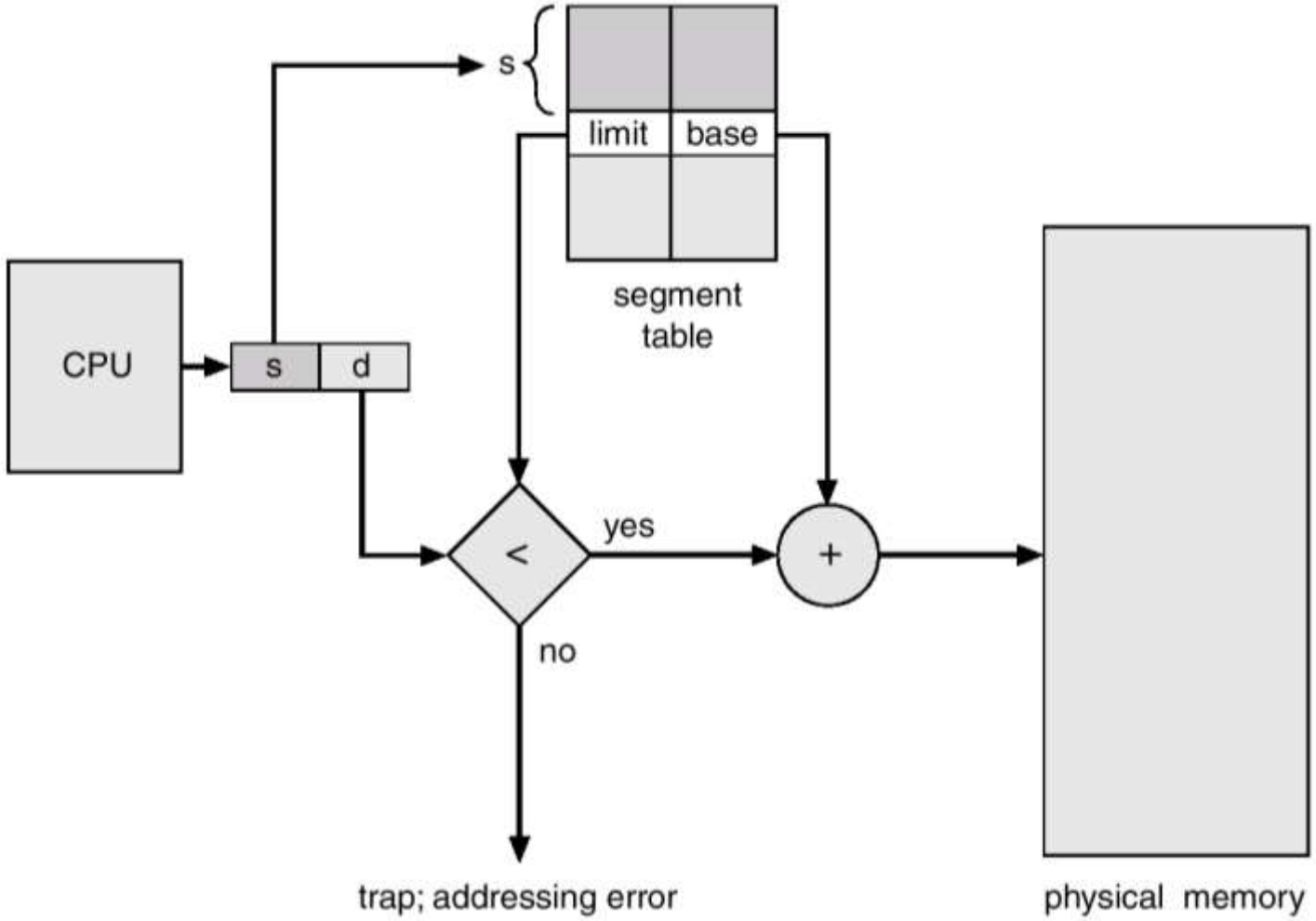
Architektura segmentacji – c.d.

- Relokacja.
  - dynamiczna
  - za pomocą tablicy segmentów
- Dzielenie dostępu.
  - dzielone segmenty
  - ten sam numer segmentu
- Alokacja.
  - first fit/best fit
  - fragmentacja zewnętrzna

Architektura segmentacji (c.d.)

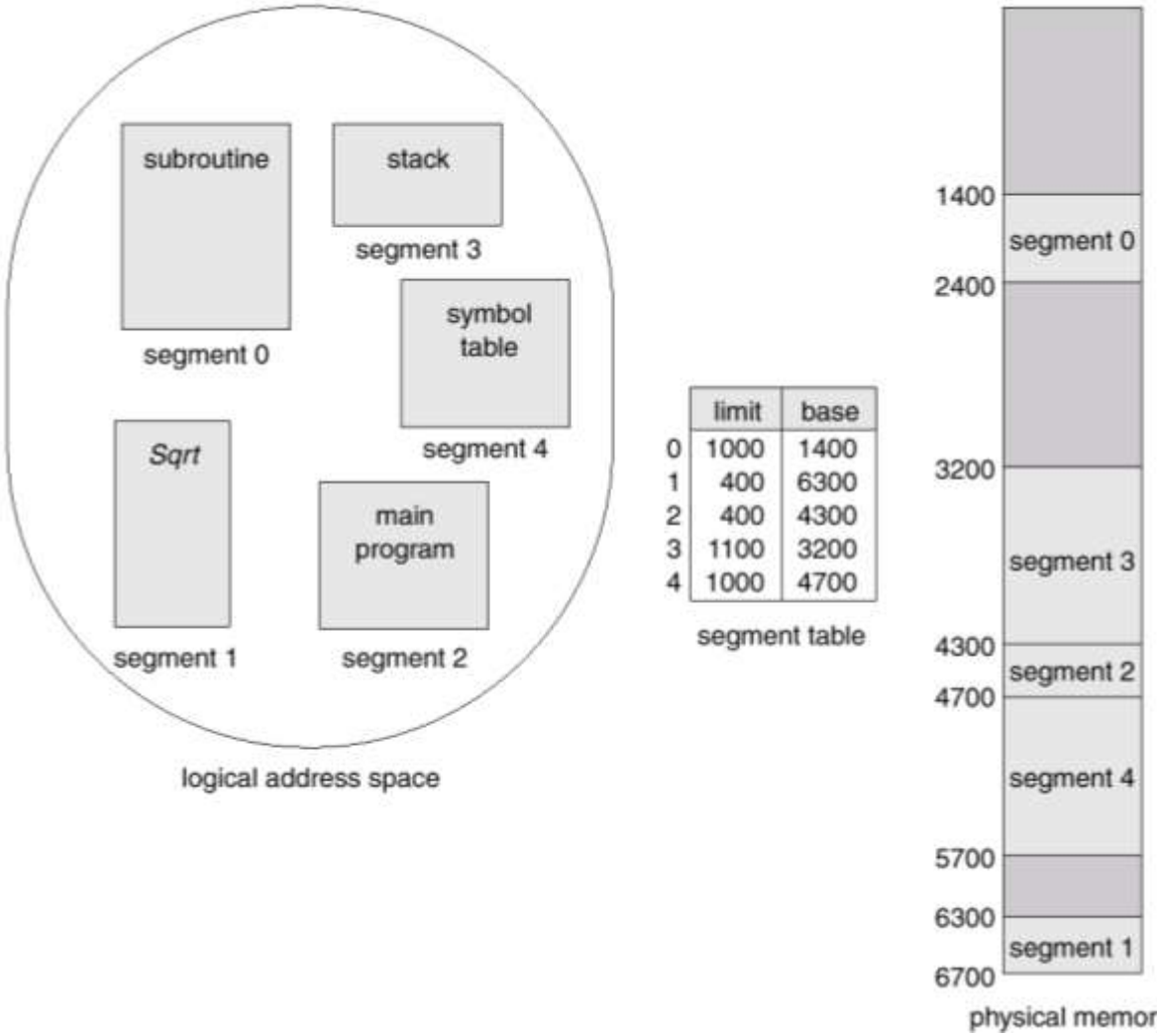
- Ochrona. Każdy wpis tablicy segmentów zawiera dodatkowo:
  - bit poprawności (= 0 $\Rightarrow$ *illegal segment*)
  - prawa *read/write/execute*
- Bity ochrony są związane z segmentami; dzielenie kodu zachodzi na poziomie segmentu.
- Ponieważ segmenty mają różne długości - powstaje problem dynamicznego przydziału pamięci.

Architektura segmentacji (c.d.)

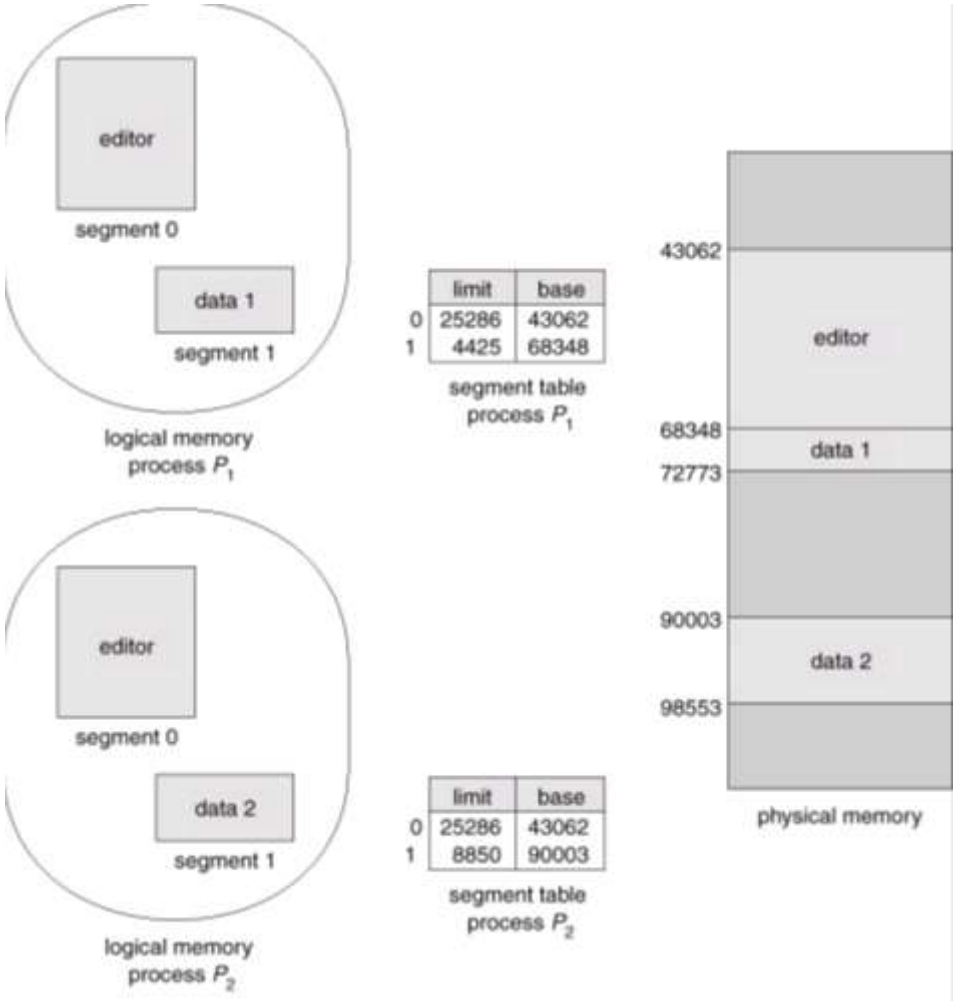


Architektura segmentacji (c.d.)

Przykład segmentacji



Przykład współdzielenia segmentów

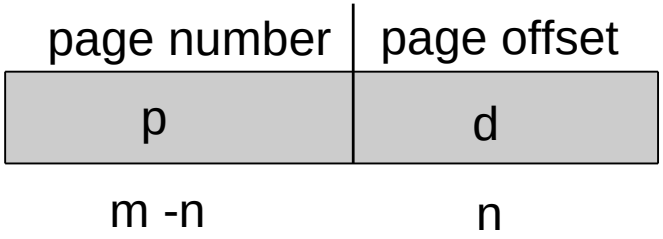


Stronicowanie

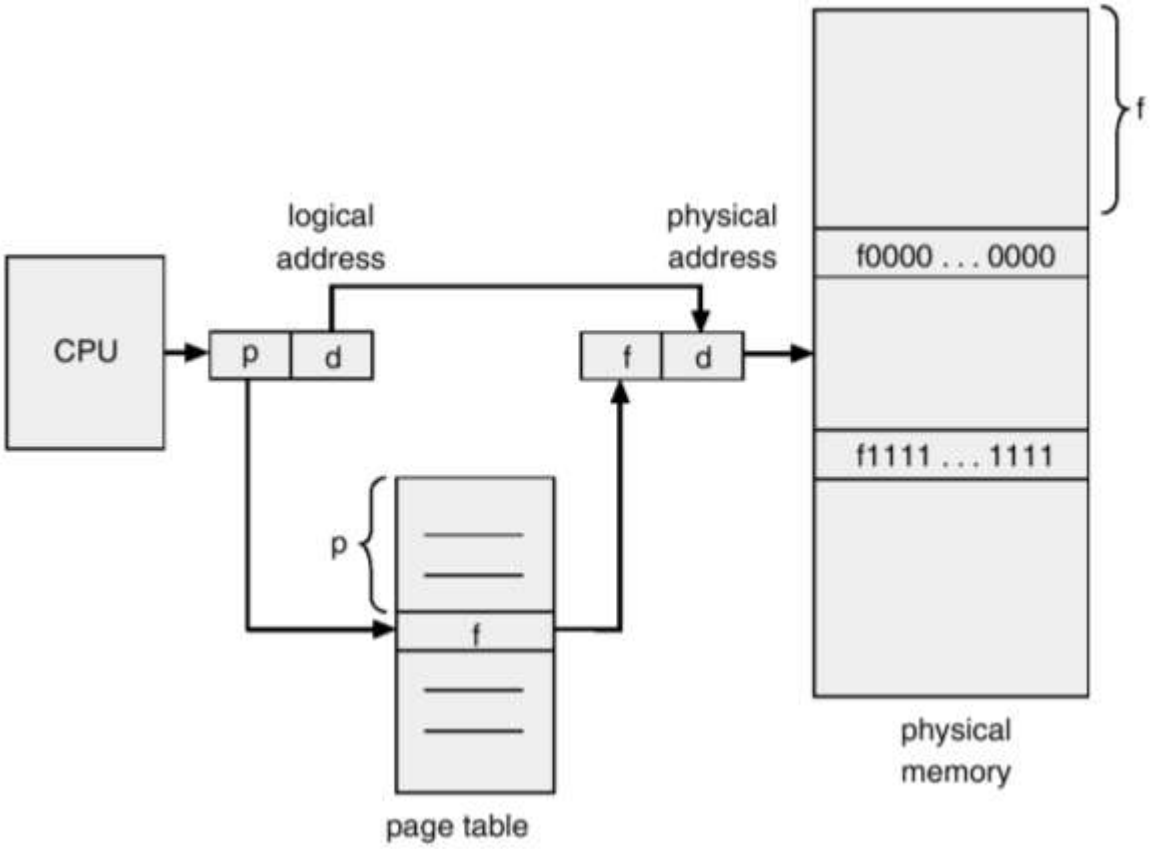
- Zbiór wykorzystywanych adresów logicznych może nie być spójny. Proces otrzymuje pamięć fizyczną w miarę dostępności.
- Pamięć fizyczna jest podzielona na bloki o stałej długości (potęga 2, pomiędzy 512 a 8192 bajtów), tzw. **ramki**.
- Przestrzeń adresów logicznych jest podzielona na bloki o tym samym rozmiarze, tzw. **strony**.
- System zarządza wszystkimi ramkami wolnymi.
- Aby wykonać program wymagający *n* stron należy znaleźć *n* wolnych ramek i wczytać program.
- System konstruuje tablicę alokacji stron, aby tłumaczyć adresy logiczne na fizyczne.
- Stronicowanie powoduje wewnętrzną fragmentację pamięci.

Tłumaczenie adresów

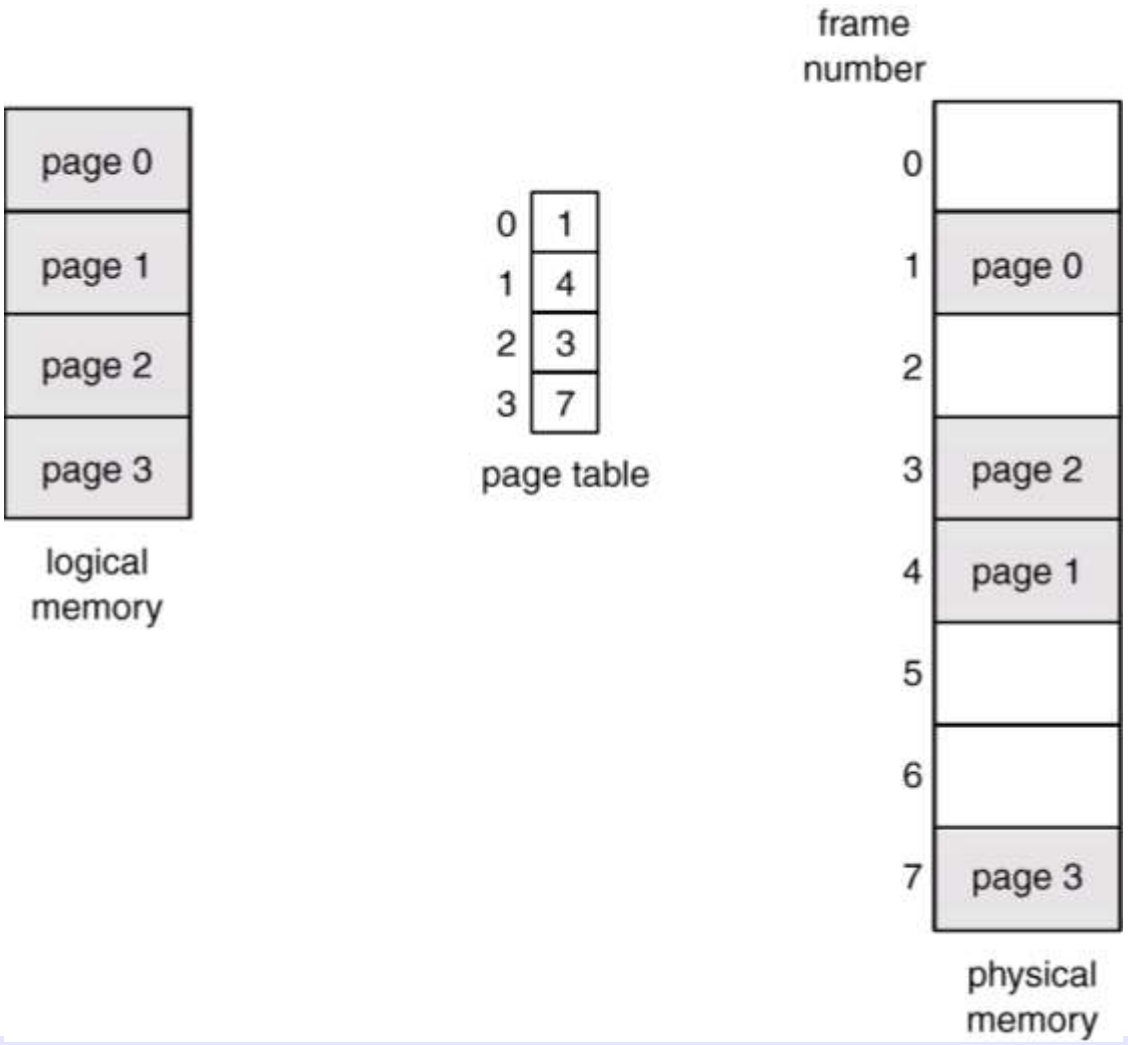
- Adres wyznaczony przez procesor dzieli się na dwie części:
  - *numer strony (p)* – używany jako indeks w *tablicy stron,* zawierającej adresy bazowe wszystkich stron w pamięci fizycznej.
  - *przesunięcie na stronie (d)* – w połączeniu z adresem bazowym definiuje adres fizyczny pamięci - przesyłany do jednostki pamięci.



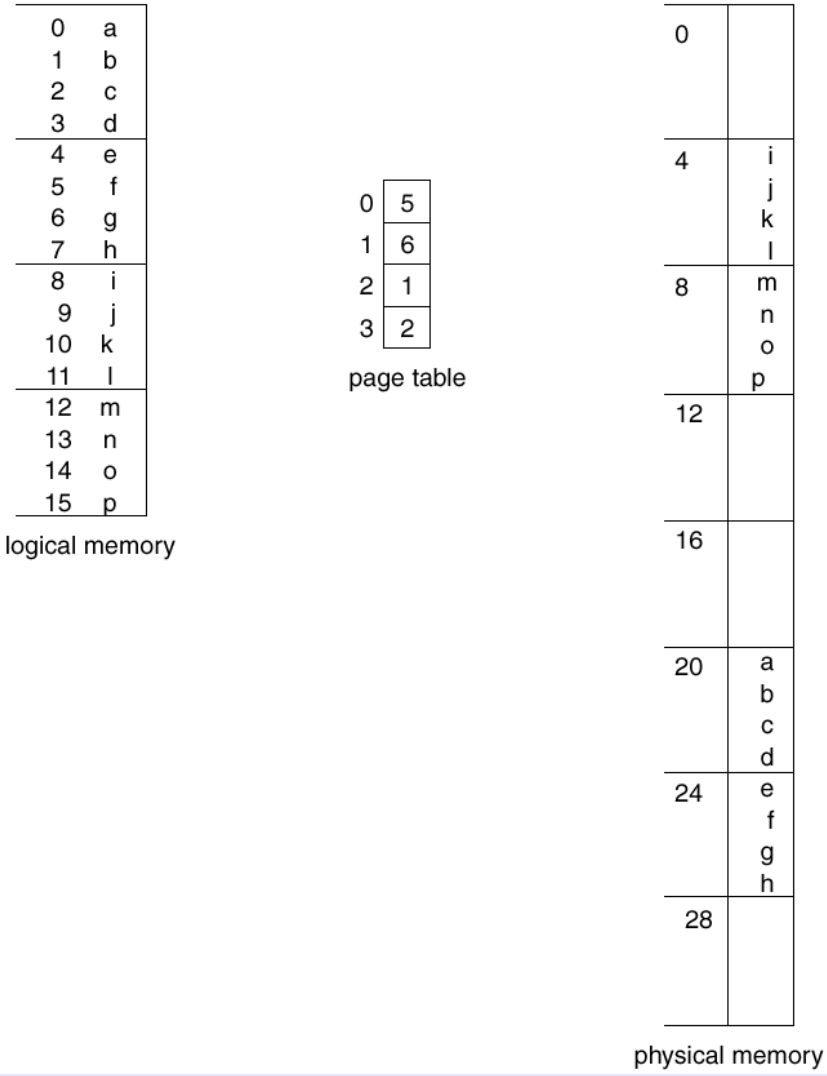
Sprzętowe tłumaczenie adresów



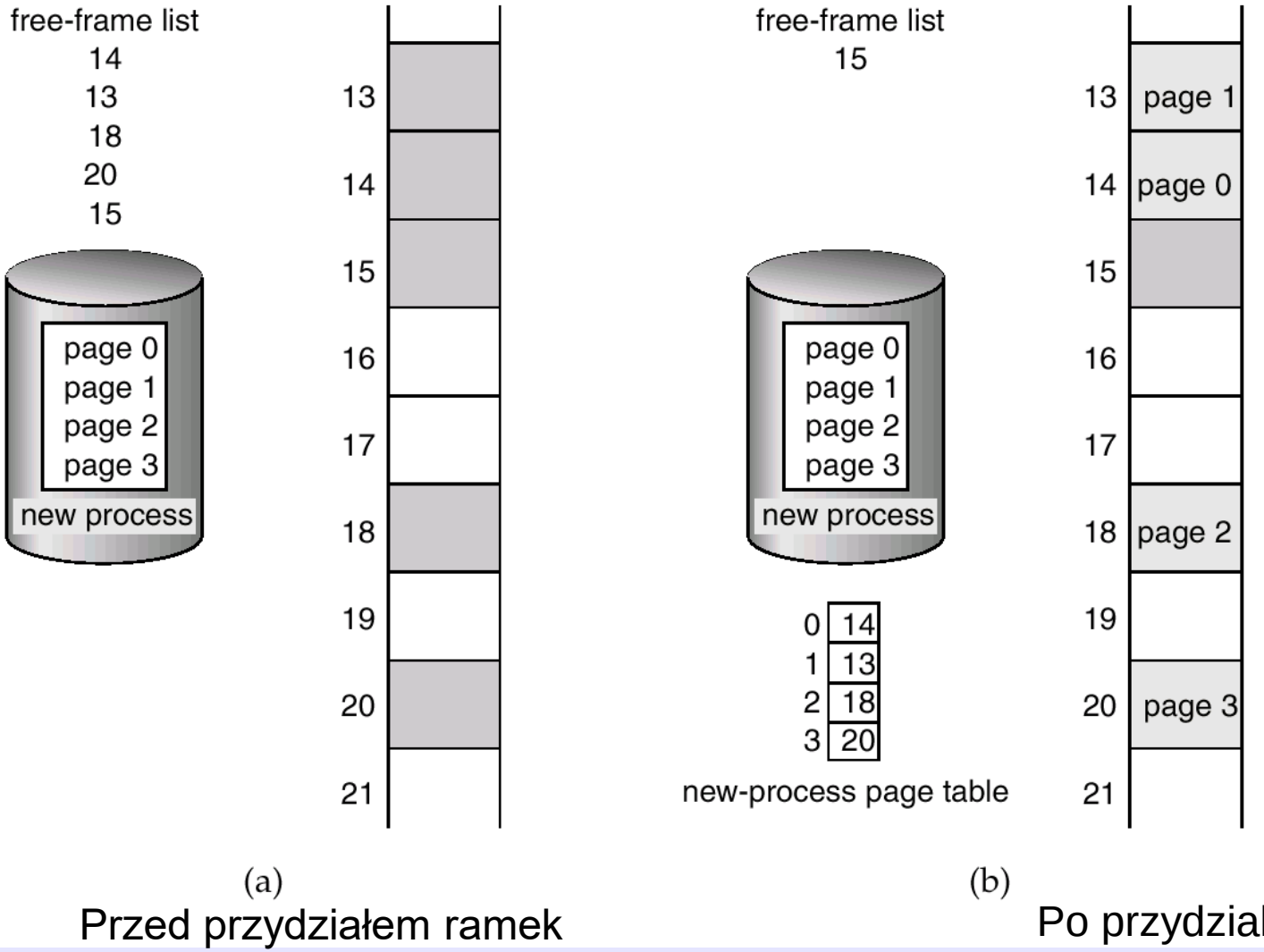
Przykład stronicowania



Przykład stronicowania



Zarządzanie ramkami wolnymi



Przed przydziałem ramek

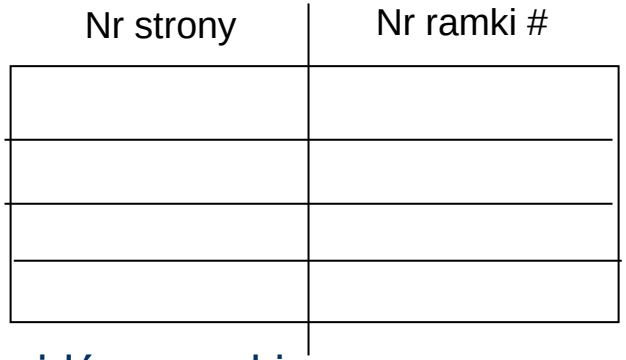
Po przydziale

Realizacje tablicy stron

- Tablica stron jest przechowywana w pamięci operacyjnej.
- *Rejestr bazowy tablicy stron (*PTBR) wskazuje położenie tablicy stron w pamięci.
- *Rejestr długości tablicy stron* (PRLR) zawiera długość tablicy stron.
- Każdy odczyt/zapis danej/kodu w pamięci wymaga dwóch dostępów do pamięci (jeden do tablicy stron, a drugi do danych/kodu).
- Standardowym rozwiązaniem problemu dwukrotnego spowolnienia dostępu do pamięci jest użycie specjalnej, małej, szybko przeszukiwanej *pamięci asocjacyjnej* (buforów translacji adresów stron, ang. *associative registers* albo *translation look-aside buffers - TLBs).*

Rejestr asocjacyjny

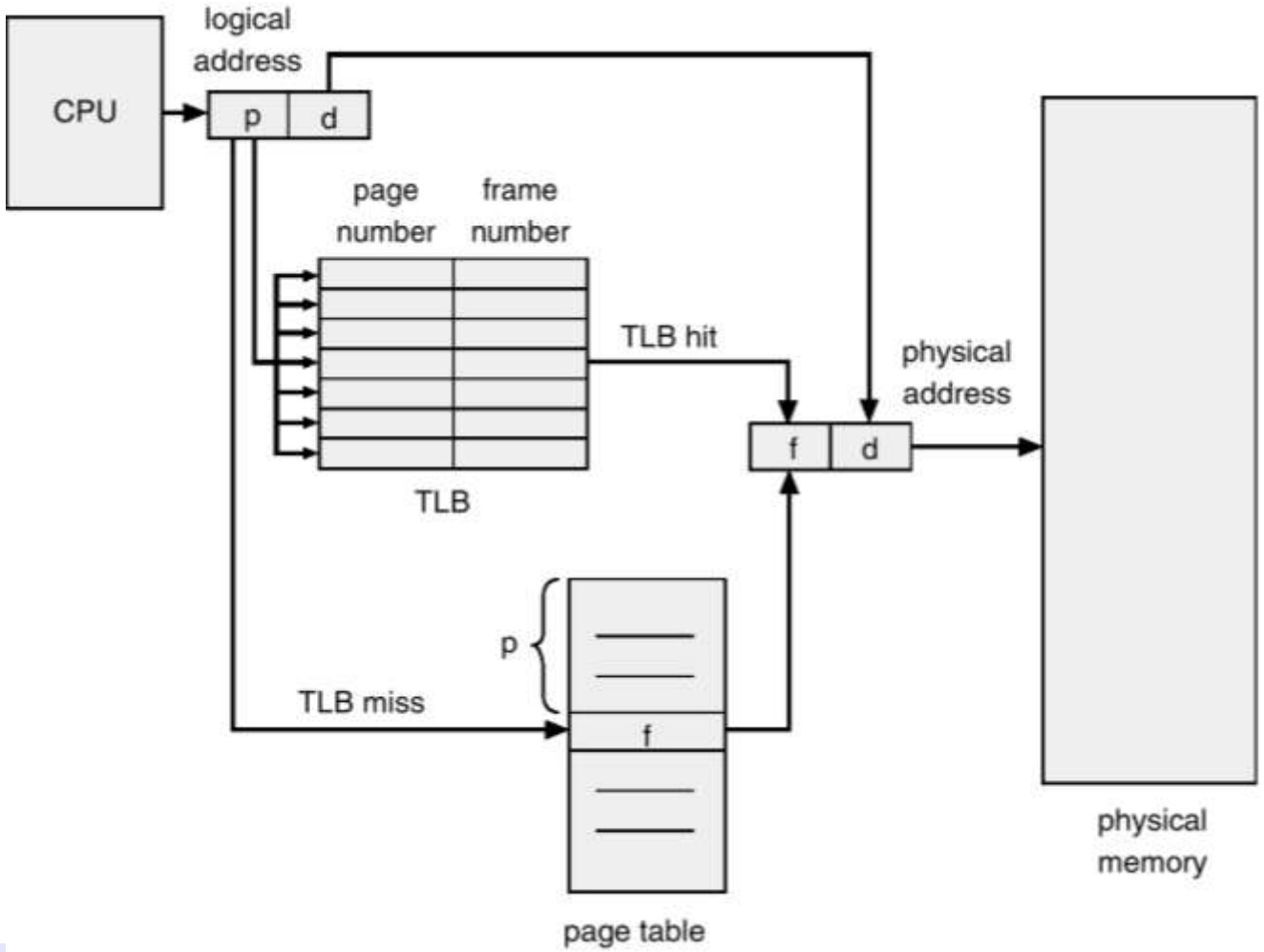
- Rejestry asocjacyjne umożliwiają równoczesne przeszukiwanie całego fragmentu tablicy stron skopiowanego do tego rejestru



- Tłumaczenie adresu (A´, A´´)
  - Jeśli A´ jest w rejestrach asocjacyjnych, to znajdź nr ramki.
  - W przeciwnym przypadku znajdź nr ramki w pełnej tablicy stron.
- Rejestry asocjacyjne mają zazwyczaj niewielki rozmiar (64 do 1024 pozycji)
- Przy niepowodzeniu szukania (*miss*) szukana wartość jest wpisywana do rejestru (na którą pozycję?)
- Niektóre rejestry przechowują identyfikatory ASID (address-space identifiers), związane z procesem którego adres dotyczy. Dzięki temu można zrealizować ochronę pamięci. W innym przypadku rejestr musi być opróżniony przy każdym przełączeniu kontekstu.

Realizacje tablicy stron – c.d.

Stronicowanie z TLB



Efektywny czas dostępu

- Czas przeszukiwania rejestrów asocjacyjnych = $\varepsilon * t_{RAM}$, gdzie t_{RAM} to czas ostępu do RAM
- Współczynnik trafień– procent numerów stron odnajdywanych w rejestrach asocjacyjnych, ozn. α
- Efektywny czas dostępu (EAT)

$$EAT = ((1 + \varepsilon)\ \alpha + (2 + \varepsilon)(1 - \alpha))* t_{RAM}$$
$$= (2 + \varepsilon - \alpha)* t_{RAM}$$

Np. dla czasu przeszukiwania TLB: 20ns i czasu dostępu do pamięci:100ns współczynnik ε=0.2. Stąd:

dla α=80% i ε=0.2, EAT=1.4

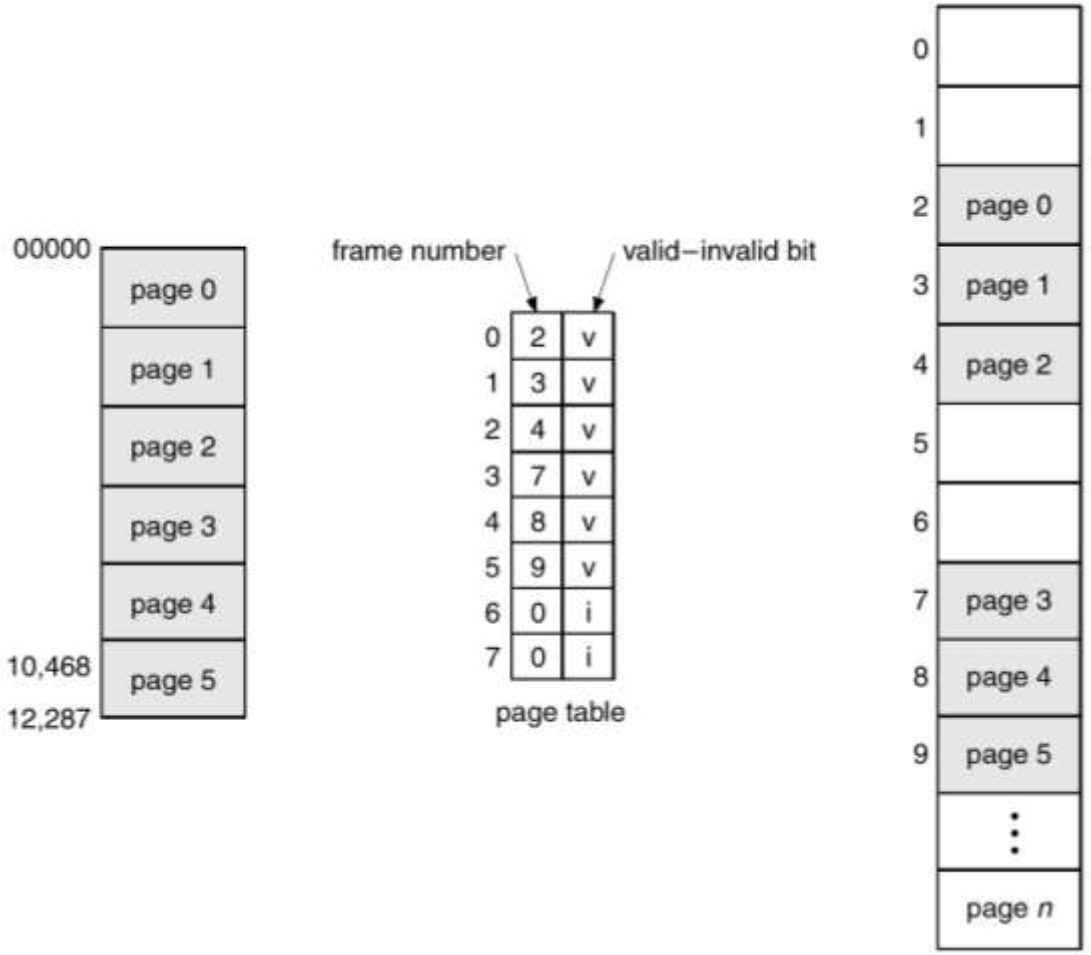
dla α=99% i ε=0.2, EAT=1.21

Ochrona pamięci

- Do ochrony pamięci w środowisku stronicowanym służą bity ochrony przypisane każdej ramce (zapobiegają zapisowi na stronach przeznaczonych tylko do odczytu).
- Dodatkowy ***bit poprawności*** w tablicy stron ma następujące znaczenie:
  - stan “poprawne” oznacza, że strona, z którą jest związany, znajduje się w logicznej przestrzeni adresowej procesu (jest poprawna).
  - Stan “niepoprawne” oznacza, że strona, z którą jest związany, nie znajduje się w logicznej przestrzeni adresowej procesu.

Ochrona pamięci – c.d.

Przykład: bity poprawności w tablicy stron

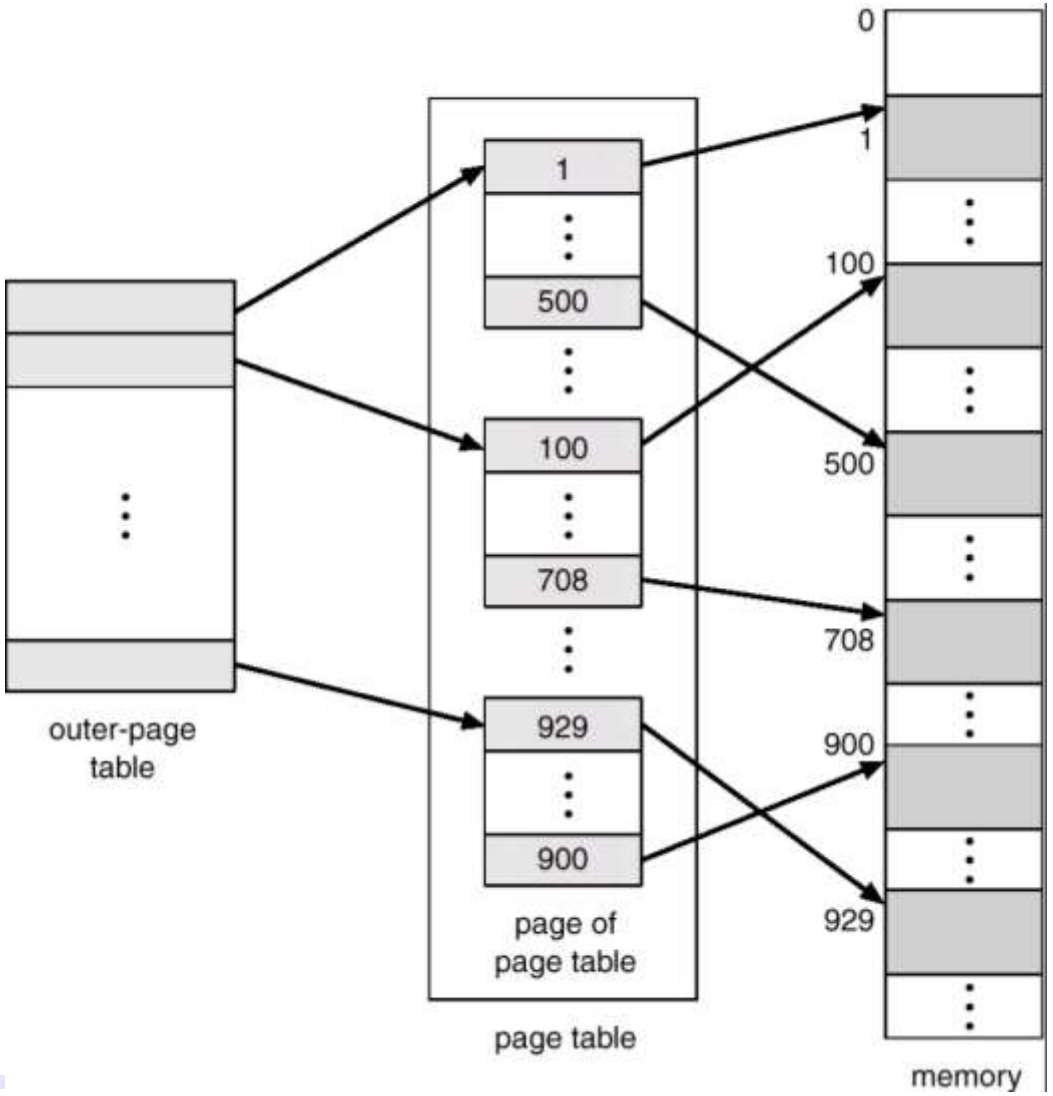


Schematy stronicowania

- Struktury danych dla stronicowania mogą osiągać duże rozmiary
  Przykład:
  - 32-bitowa przestrzeń adresów logicznych
  - Rozmiar strony: 4 KB (2^{12})
  - Tablica stron ma ponad 1 milion pozycji(2^{32} / 2^{12})
  - Jeśli każda pozycja tablicy zajmuje 4 bajty -> tablica stron zajmuje 4 MB pamięci fizycznej
- Rozwiązania:
  - Stronicowanie hierarchiczne
  - Stronicowanie z tablicą rzadką (*hashed page table*)
  - Odwrócone tablice stron

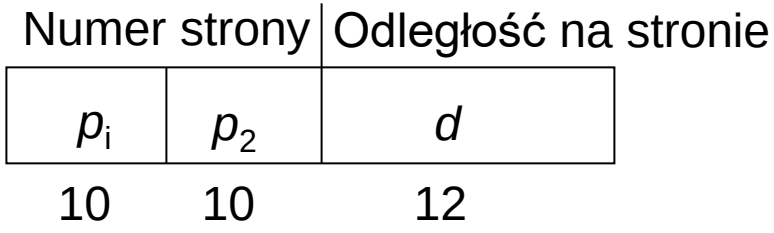
Stronicowanie hierarchiczne

- Podział przestrzeni adresów logicznych pomiędzy wiele tablic stron.
- Przykład obok: stronicowanie z dwupoziomową strukturą tablic stron.



Przykład stronicowania dwupoziomowego

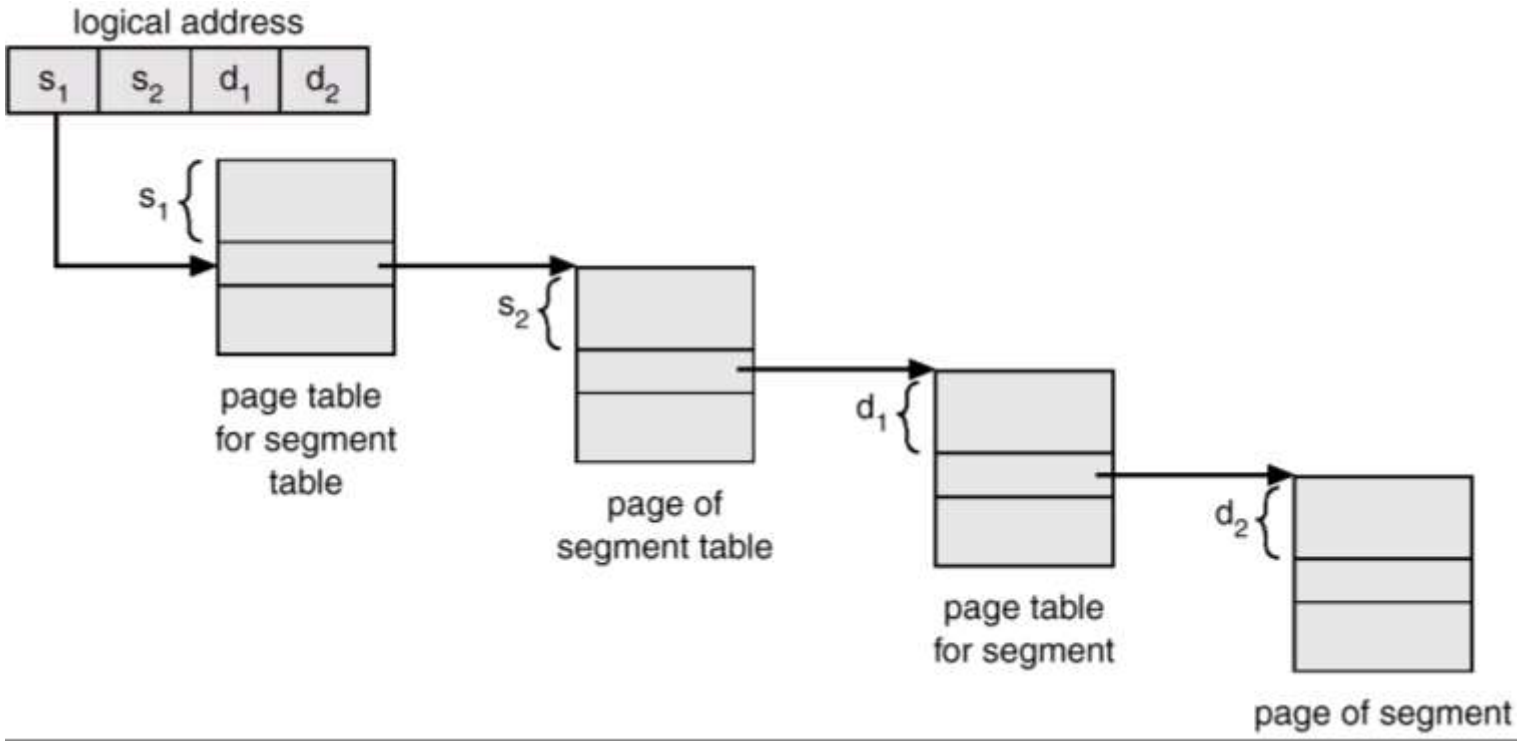
- Adres logiczny na 32-bitowej maszynie o 4K stronie jest podzielony na:
  - 20-bitowy numer strony
  - 12-bitową odległość na stronie.
- Ponieważ tablica stron jest stronicowana, to numer strony jest dalej dzielony na:
  - 10-bitowy numer strony
  - 10-bitowe przesunięcie na stronie.
- Adres logiczny przyjmuje więc postać:



gdzie p_i jest indeksem do zewnętrznej tablicy stron, a p_2 jest przesunięciem w zewnętrznej tablicy stron.(**forward-mapped page table**)

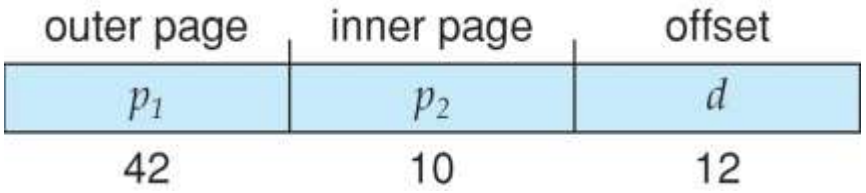
Tłumaczenie adresu

- Tłumaczenie adresu w dwupoziomowej architekturze 32-bitowej

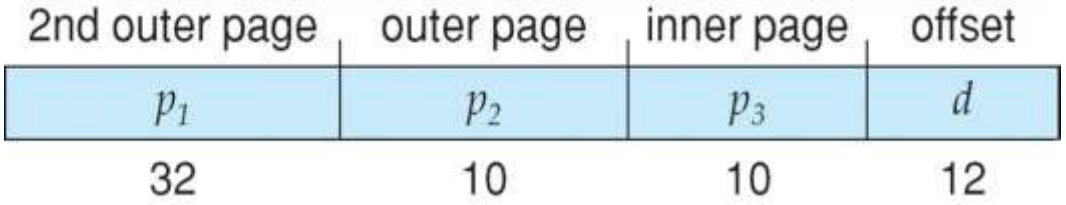


64-bitowa przestrzeń adresów logicznych

- Dla 64 b dwupoziomowe stronicowanie nie wystarcza
- Dla strony o rozmiarze 4 KB (2^{12})
  - Tablica stron ma 2^{52} pozycji
  - W schemacie dwupoziomowym tablica wewnętrzna zawiera 2^{10} 4-bajtowych wpisów
  - Adres ma postać



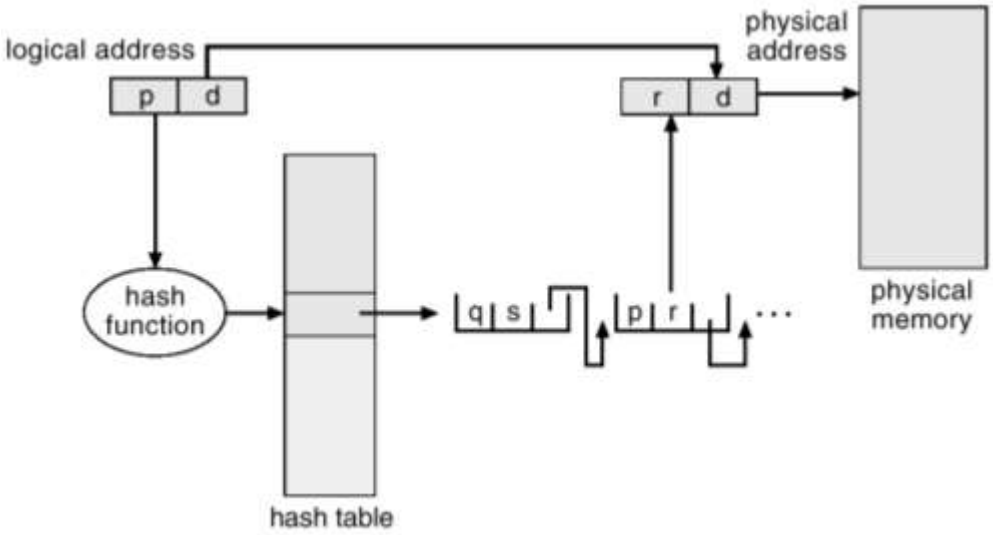
  - Tablica zewnętrzna ma 2^{42} pozycji, tzn. rozmiar aż 2^{44} bajtów
  - Jedno z rozwiązań: druga tablica zewnętrzna (schemat adresowania trzypoziomowego).



  - Druga tablica zewnętrzna jest nadal duża: zawiera 2^{34} bajtów. Ponadto potrzeba 4-ch dostępów do pamięci by sięgnąć do fizycznej komórki pamięci.

Stronicowanie z tablicą rzadką

- Schemat popularny dla adresów > 32 bitów.
- Numer wirtualnej strony jest odwzorowany za pomocą **funkcji mieszającej** (haszującej) w tablicę stron. Elementy tablicy stron wskazują na listy par (nr strona, nr ramki) odwzorowanych w to samo miejsce tablicy stron.
- Numer wirtualnej strony jest porównywany z kolejnymi elementami listy. Jeśli znaleziono właściwy numer – z elementu struktury listy pobierany jest numer ramki.

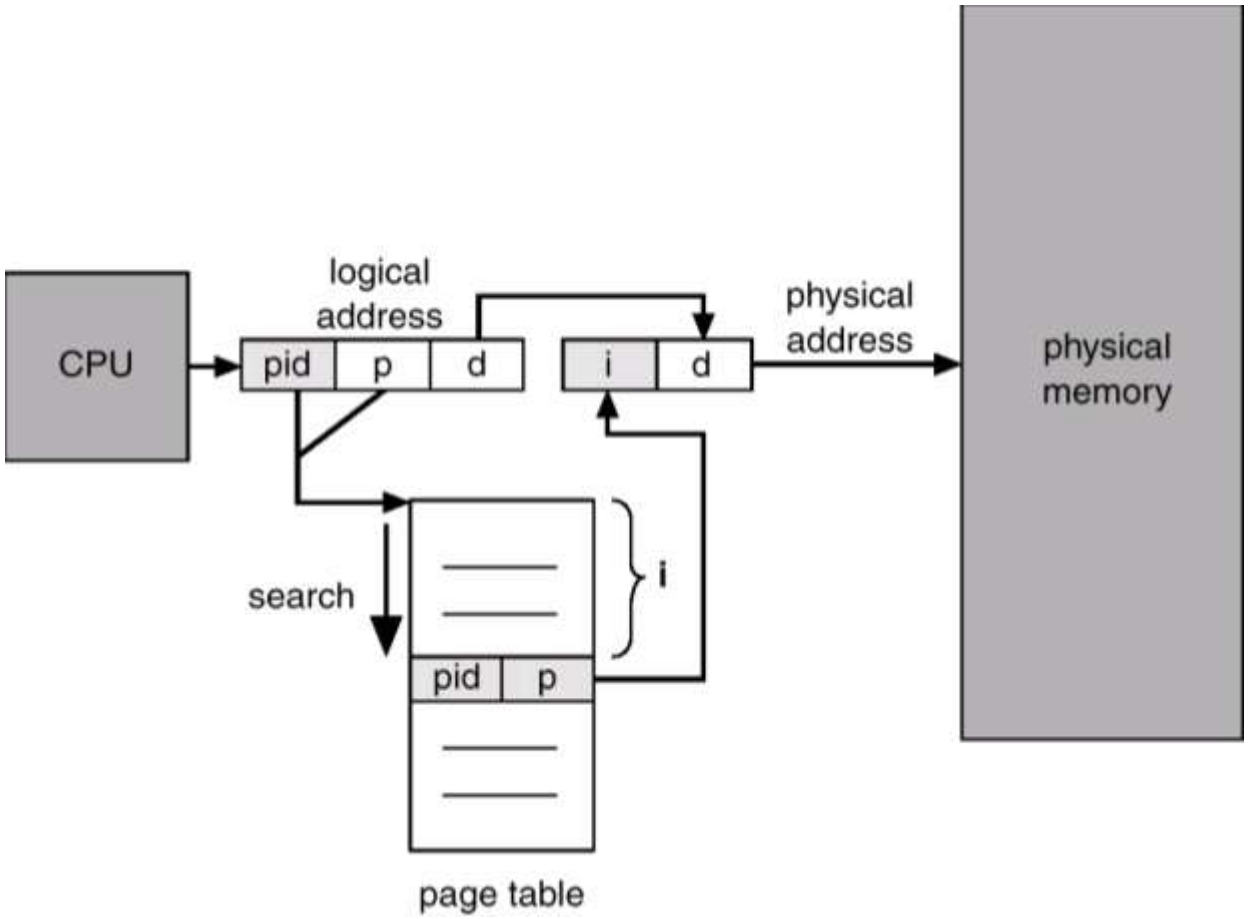


- Odmiana schematu - grupowe tablice stron (**clustered page tables**)
  - Każda pozycja tablicy dotyczy kilku stron (np. 16) a nie jednej
  - Takie tablice są szczególnie użyteczne dla rzadkich (**sparse**) przestrzeni adresowych (wykorzystuje się szereg podzbiorów adresów)

Odwrócona tablica stron

- Odwrócona tablica stron ma po jednym wpisie dla każdej strony pamięci fizycznej (ramki).
- Wpis zawiera wirtualny adres strony przechowywanej w pamięci operacyjnej oraz informację o procesie-właścicielu strony.
- Odwrócona tablica stron zmniejsza zajętość pamięci przez tablice stron, ale zwiększa czas przeszukiwania tablicy przy odwołaniu do strony.
- Dla zmniejszenia czasu przeszukiwania wprowadza się tablicę haszowania, umożliwiająca ograniczenie szukania do najwyżej kilku wpisów z tablicy stron.

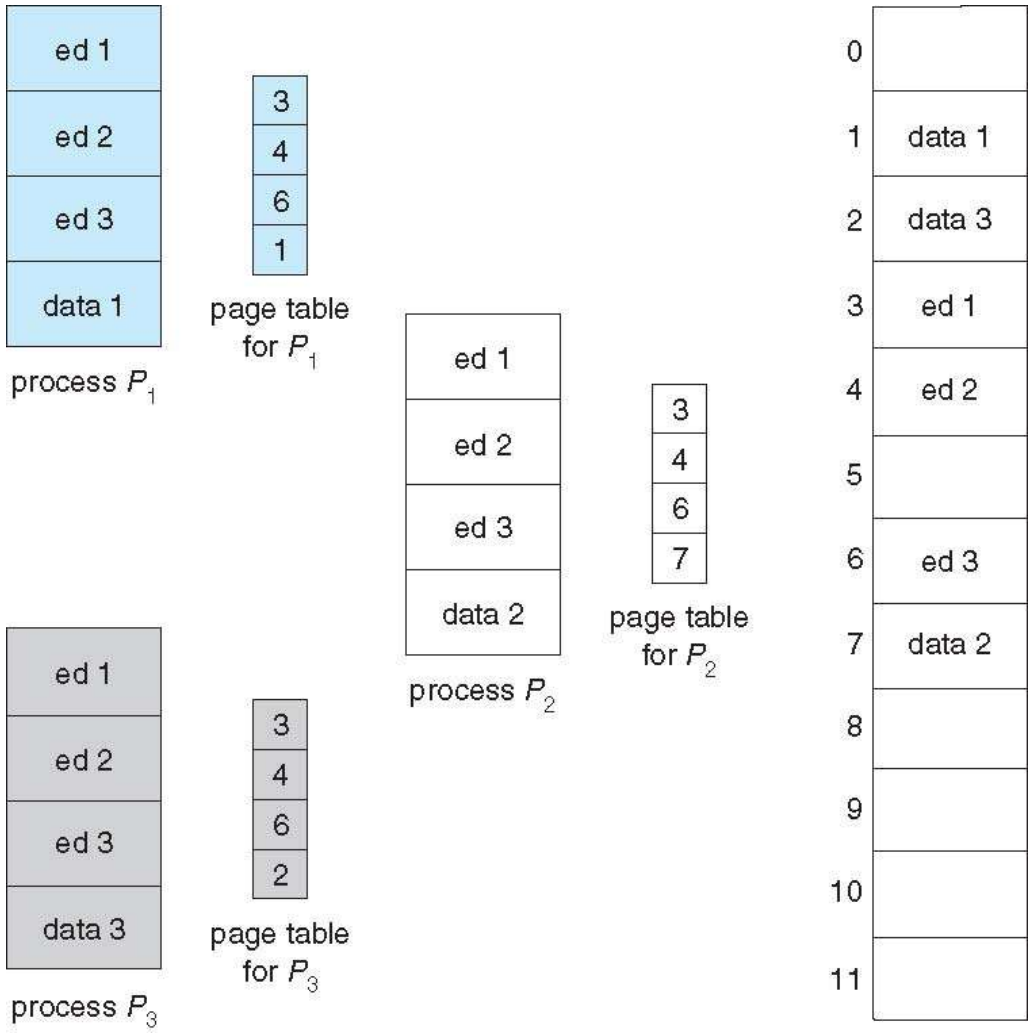
Odwrócona tablica stron



Strony współdzielone

- Dzielenie kodu w środowisku stronicowanym
  - Jeden egzemplarz kodu wznawialnego (czystego, *reentrant*) może być dzielony przez wiele procesów (np. edytory tekstu, kompilatory, system okien).
  - Dzielony kod musi być odwzorowany w ten sam adres logiczny dla wszystkich procesów.
- Prywatny kod i dane
  - Każdy proces ma swoją kopię rejestrów i obszar danych podczas działania.
  - Strony prywatnych danych są odwzorowywane na różne ramki.
- Implementowanie pamięci dzielonej w systemach z odwróconą tablicą stron napotyka na trudności (potrzeba odwzorowywać wiele adresów wirtualnych na jeden adres fizyczny)

Przykład współdzielenia stron procesu



Example: The Intel 32 and 64-bit Architectures (*)

- Dominant industry chips
- Pentium CPUs are 32-bit and called IA-32 architecture
- Current Intel CPUs are 64-bit and called IA-64 architecture
- Many variations in the chips, cover the main ideas here

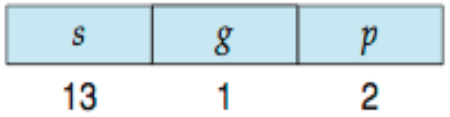
(*) materiał dodatkowy

Example: The Intel IA-32 Architecture

- Supports both segmentation and segmentation with paging
- Each segment can be 4 GB
- Up to 16 K segments per process
- Divided into two partitions
  - First partition of up to 8 K segments are private to process (kept in **local descriptor table** (**LDT**))
  - Second partition of up to 8K segments shared among all processes (kept in **global descriptor table** (**GDT**))

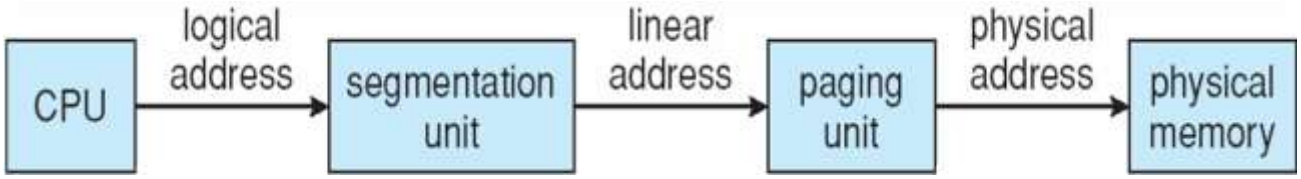
Example: The Intel IA-32 Architecture (Cont.)

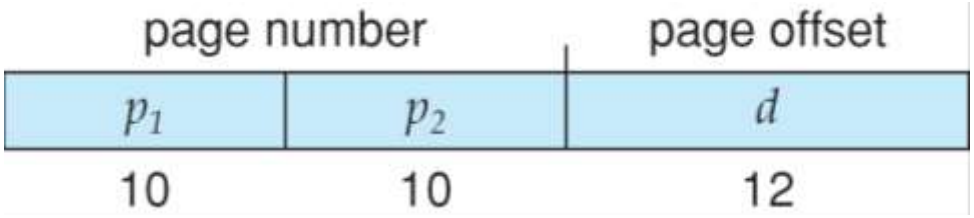
- CPU generates logical address
  - Selector given to segmentation unit
    - Which produces linear addresses



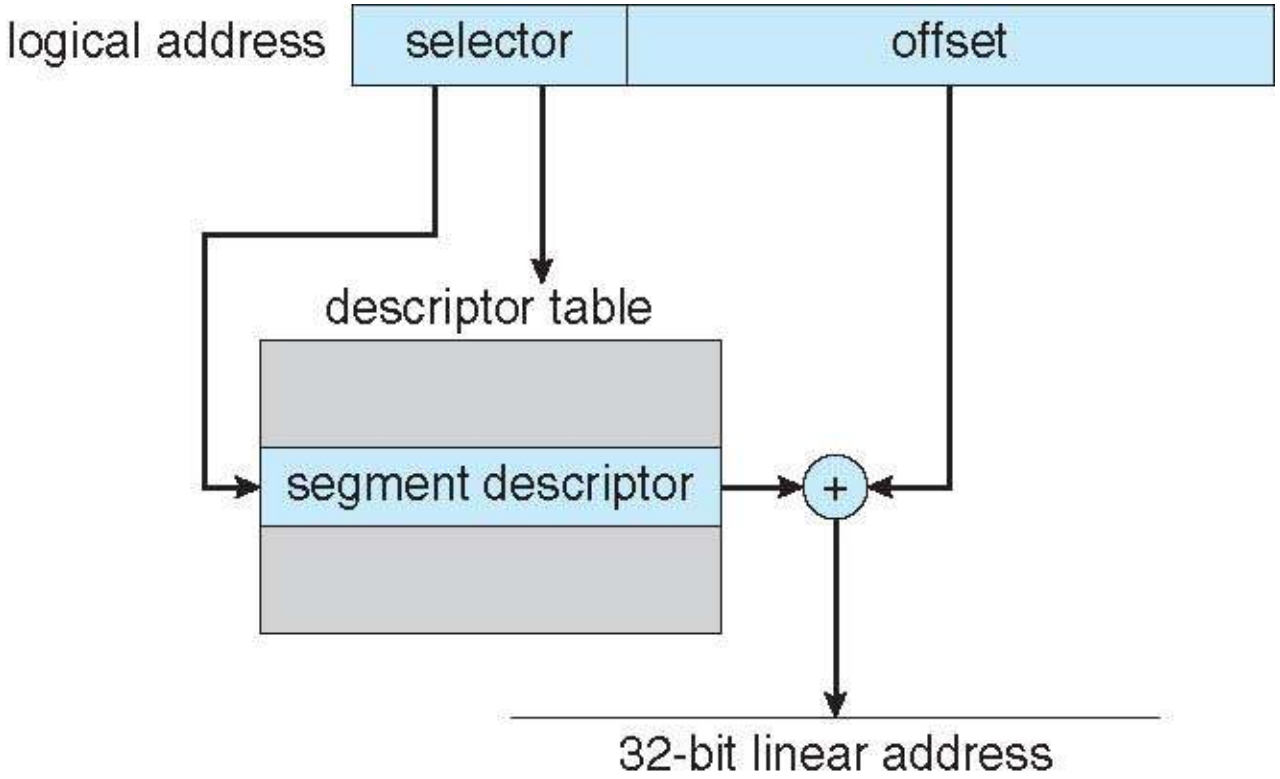
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
    - Pages sizes can be 4 KB or 4 MB

Logical to Physical Address Translation in IA-32

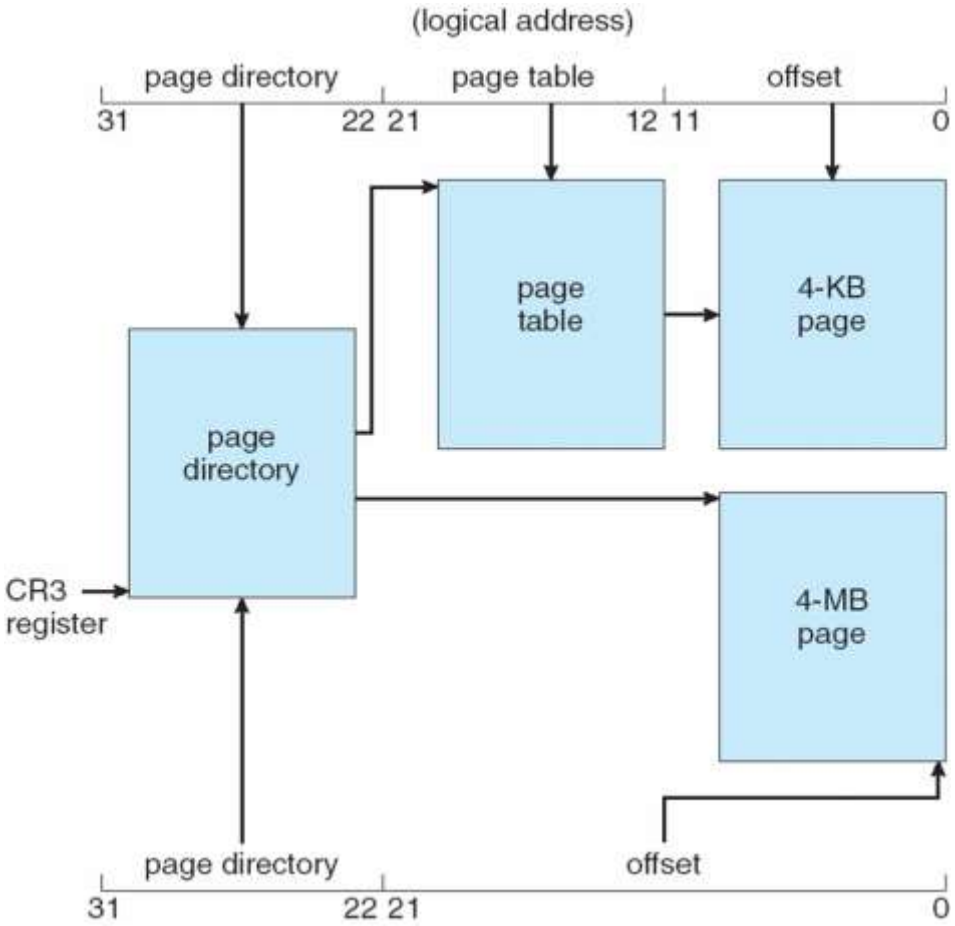




Intel IA-32 Segmentation

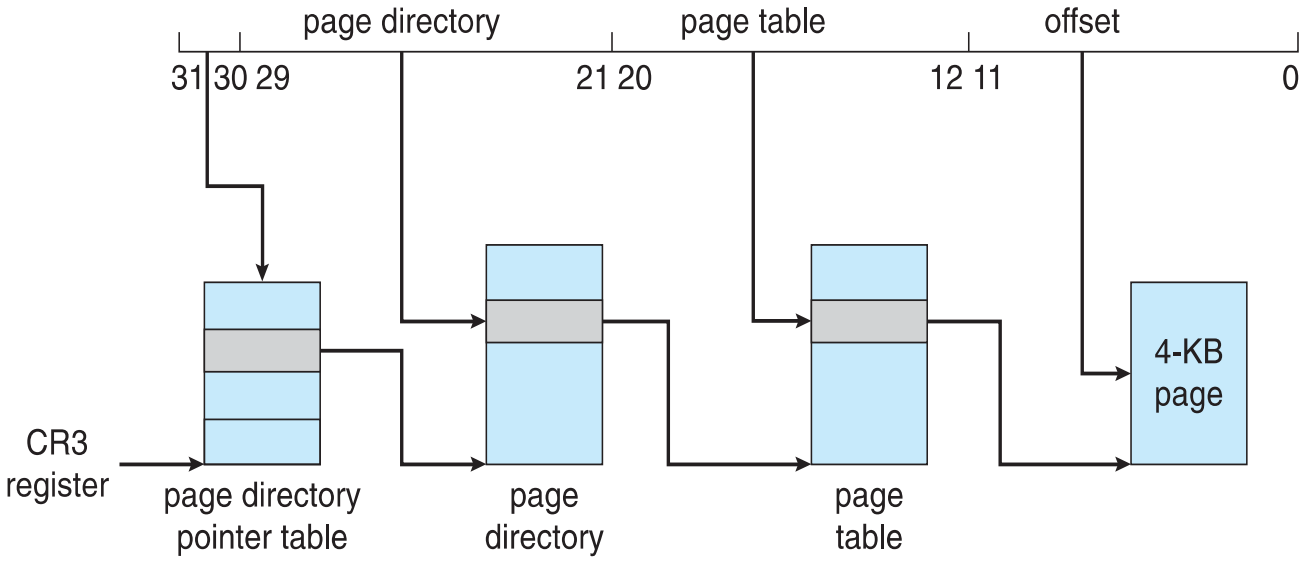


Intel IA-32 Paging Architecture



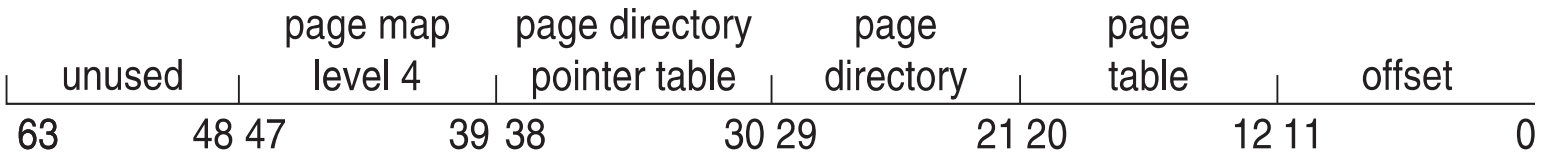
Intel IA-32 Page Address Extensions

- 32-bit address limits led Intel to create **page address extension** (**PAE**), allowing 32-bit apps access to more than 4GB of memory space
  - Paging went to a 3-level scheme
  - Top two bits refer to a **page directory pointer table**
  - Page-directory and page-table entries moved to 64-bits in size
  - Net effect is increasing address space to 36 bits – 64GB of physical memory



Intel x86-64

- Current generation Intel x86 architecture
- 64 bits make enormous address space (> 16 exabytes)
- In practice only implement 48 bit addressing
  - Page sizes of 4 KB, 2 MB, 1 GB
  - Four levels of paging hierarchy
- Can also use PAE so virtual addresses are 48 bits and physical addresses are 52 bits



Example: ARM Architecture

- Dominant mobile platform chip (Apple iOS and Google Android devices for example)
- Modern, energy efficient, 32-bit CPU
- 4 KB and 16 KB pages
- 1 MB and 16 MB pages (termed **sections**)
- One-level paging for sections, two-level for smaller pages
- Two levels of TLBs
  - Outer level has two micro TLBs (one data, one instruction)
  - Inner is single main TLB
  - First inner is checked, on miss outers are checked, and on miss page table walk performed by CPU