# Zarządzanie pamięcią

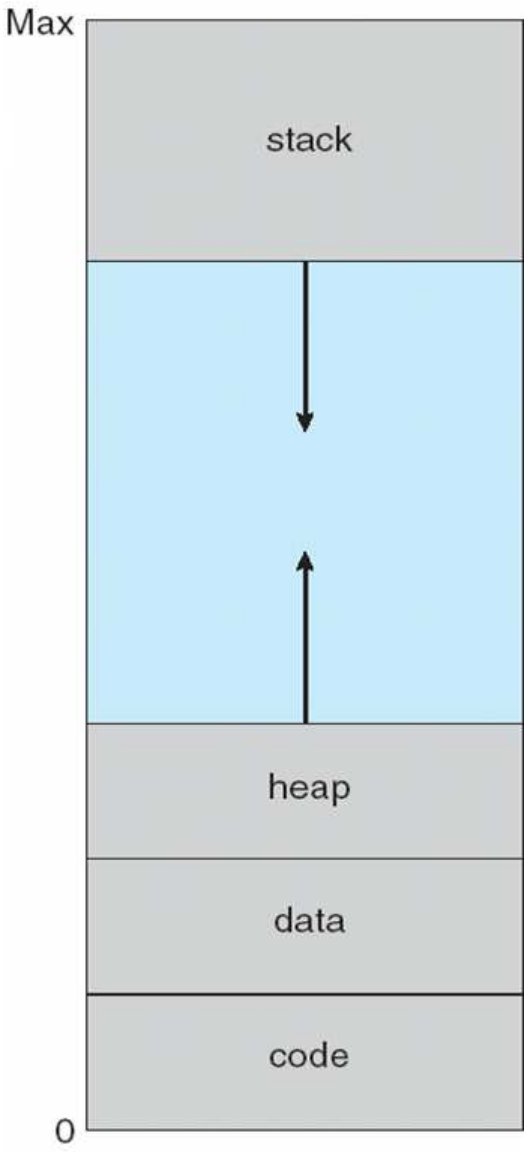
W dawnych systemach operacyjnych, ale także w bardziej prymitywnych systemach współczesnych (np. w małych systemach przeznaczonych na platformy wbudowane), operacje zarządzania pamięcią, takie jak jej przydzielanie, wykorzystanie, i zwalnianie, było pozostawione procesom lub programom. W nowoczesnych i większych systemach operacyjnych coraz więcej funkcji zarządzania pamięcią obsługuje system operacyjny, ze wsparciem od strony sprzętu, pozostawiając procesom swobodę wykorzystania pamięci.

Funkcje zarządzania pamięcią:

- przydział obszaru pamięci dla procesu, i w tym celu: stronicowanie, wymiatanie i przywracanie procesów,
- translacja adresów wirtualnych,
- ochrona pamięci — zapewnienie by programy odwoływały się jedynie do swoich własnych obszarów pamięci.

Zarządzaniem pamięcią zajmuje się jądro systemu, z wykorzystaniem sprzętowej jednostki zarządzania pamięcią MMU (*Memory Management Unit*).

# Wykorzystanie pamięci operacyjnej przez proces



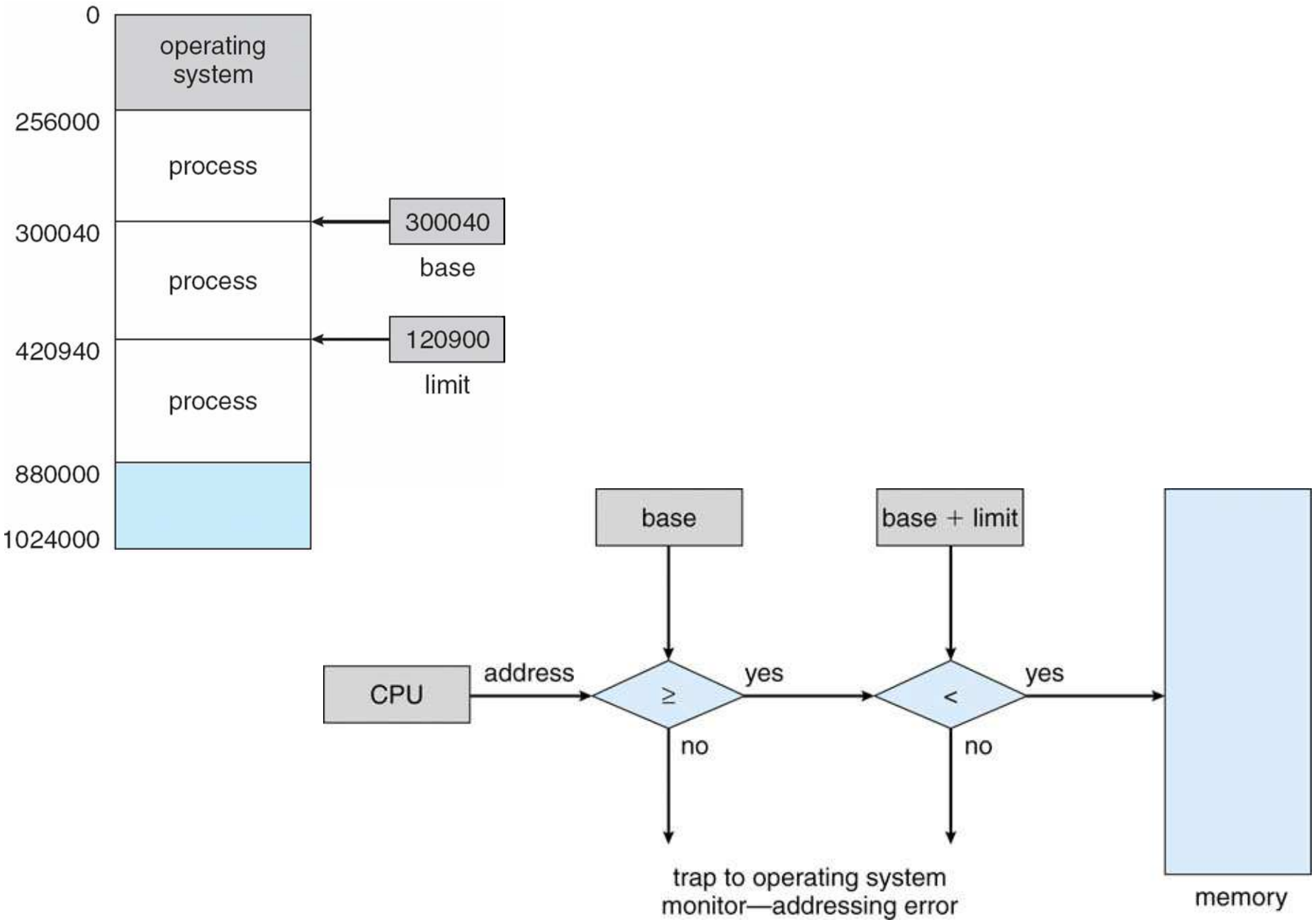

Ten prosty model wykorzystania pamięci jest atrakcyjny, ponieważ nie wymaga podziału wolnej przestrzeni na stos i stertę. Przynajmniej teoretycznie, jedno i drugie może alokować wolną pamięć aż do jej całkowitego wyczerpania.

Jednak nie do końca. Obsługa stosu (*stack*) jest zgodna z tzw. **dyscypliną stosową** (*stack discipline*). Wynika ona z tego, że procedury kończą pracę w kolejności odwrotnej do ich wywoływania. Zatem blok pamięci przydzielany dla wywoływanej procedury (zawierający jej adres powrotu, argumenty, i zmienne lokalne) będzie „przykryty" na stosie przez bloki następnie wywoływanych procedur. Jednak w momencie jej zakończenia, inne procedury wywołane w trakcie jej wykonywania będą już wcześniej zakończone, i ich pamięć na stosie zwolniona. W czasie pracy programu stos przyrasta i skraca się, ale nie powstają w nim dziury.

Tej dyscypliny nie ma sterta (*heap*), czyli pamięć arbitralnie przydzielana i zwalniana w trakcie wykonywania programu.

W programie intensywnie przydzielającym pamięć dynamiczną (funkcją `malloc`) sterta może „podziurawić" wolną przestrzeń, utrudniając jej obsługę.

# Wykorzystanie pamięci operacyjnej przez system



# Wiązanie danych i instrukcji z adresami

Związanie instrukcji i danych z adresami pamięci może następować na trzech etapach tworzenia procesu:

- na etapie kompilacji, co jest możliwe tylko w przypadku znajomości adresu początkowego programu w pamięci, i wtedy wszystkie adresy w programie mogą być bezwzględne, ale w przypadku przeniesienia programu na inne miejsce w pamięci konieczna jest rekompilacja,

- na etapie ładowania, gdzie kod programu musi być **przemieszczalny**, tzn. zawierać wyłącznie adresy względne, i wtedy program może być przenoszony w różne miejsca pamięci, a wszystkie adresy są obliczane względem adresu początkowego

- na etapie wykonywania, co pozwala przenosić kod programu w czasie pracy, ale wymaga wsparcia sprzętowego dla przeliczania wszystkich adresów względem adresu początkowego każdego segmentu pamięci.

# Adresy logiczne i fizyczne

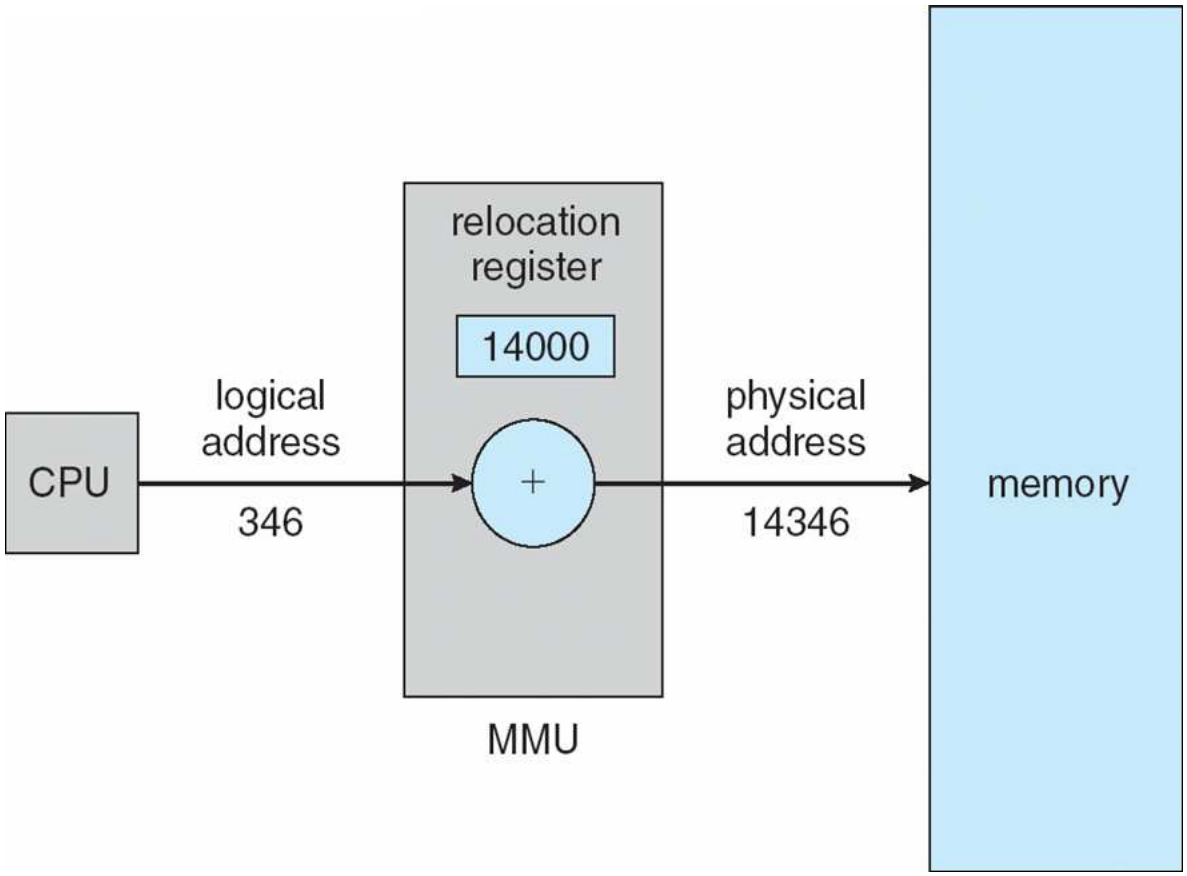
**Adres logiczny** pamięci, również zwany **adresem wirtualnym**, to adres jakim posługuje się proces. Adresy logiczne są obliczane w trakcie wykonywania procesu.

**Adres fizyczny** jest to adres komórki pamięci podawany na szynę adresową pamięci w celu pobrania lub zapisania wartości.

Proces zamiany adresów logicznych na fizyczne nazywa się **translacją adresów**. Każde odwołanie do pamięci wymaga translacji adresu.

Obserwacja: w przypadkach wiązania adresów na etapie kompilacji, adresy logiczne są identyczne z adresami fizycznymi, i translacja adresów jest trywialna. W przypadku wiązania na etapie ładowania algorytm translacji jest również dość prosty.

# Prosty mechanizm translacji adresów




Powyższy mechanizm translacji adresów zapewniany przez uproszczoną jednostkę MMU jest adekwatny dla modelu wiązania w czasie ładowania.

# Metody alokacji ciągłej

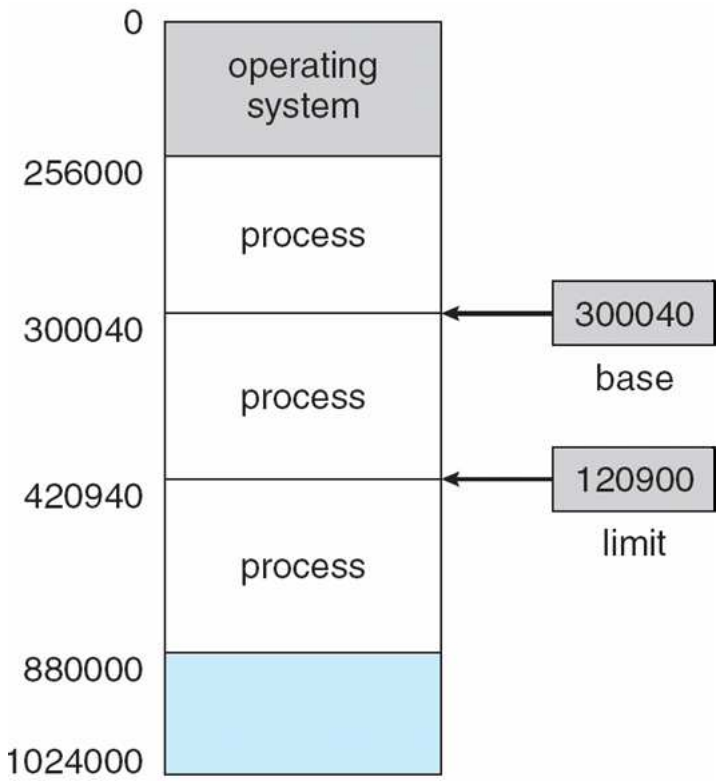
Najprostsze metody alokacji pamięci przydzielają procesowi/om, jak również systemowi operacyjnemu, wydzielony na stałe nieprzerwany blok pamięci. W ogólnym przypadku możliwe jest wiązanie adresów najwcześniej na etapie ładowania.

W **jednoprogramowych systemach operacyjnych** ta metoda sprowadza się do wydzielenia obszaru dla systemu operacyjnego i obszaru dla programu.

**Systemy wieloprogramowe** wymagają podziału pamięci na wiele bloków i ich przydział różnym programom. Podział ten może być statyczny lub dynamiczny. Metody przydziału bloków programom:

- pierwsze dopasowanie (*first fit*)
- najlepsze dopasowanie (*best fit*)
- najgorsze dopasowanie (*worst fit*)

Pamiętanie adresów: bazowego i granicznego każdego programu umożliwia ochronę pamięci.



# Fragmentacja

W trakcie pracy systemu z metodą ciągłej alokacji, pamięć ulega **fragmentacji**, która jest niekorzystnym zjawiskiem zmniejszającym efektywność przydziału. Można rozróżnić dwa rodzaje fragmentacji:

- fragmentacja wewnętrzna, wynikająca z przydzielania bloków w ustalonych jednostkach alokacji, np. 2,4,8, lub 16 kB, zawsze większych niż niezbędne procesom,

- fragmentacja zewnętrzna, wynikająca z otrzymania w trakcie pracy systemu wielu mniejszych lecz rozłącznych bloków, co może uniemożliwić przydział pamięci procesowi żądającemu większego bloku.

Fragmentację wewnętrzną można ograniczać stosując małe jednostki alokacji, co jednak zmniejsza efektywność operowania blokami pamięci.

Fragmentację zewnętrzną można eliminować przez defragmentację, możliwą jedynie w przypadku wiązania co najmniej na etapie łączenia.

# Krótkie podsumowanie — pytania sprawdzające

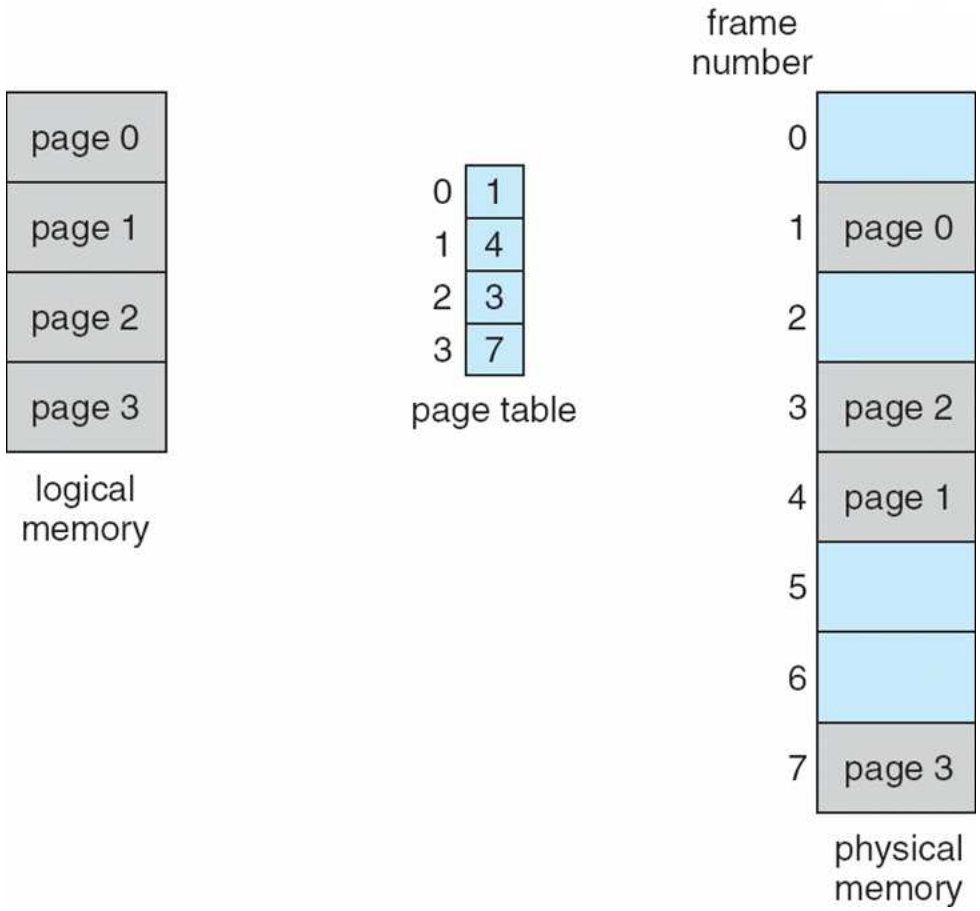
1. Jakie funkcje zarządzania pamięcią realizuje system operacyjny?
2. Jakie są możliwości wiązania danych i instrukcji z adresami?
3. Czym różnią się adresy logiczne od fizycznych?
4. Jakie są stosowane algorytmy ciągłej alokacji pamięci?
5. Na czym polega proces translacji adresu w systemie alokacji ciągłej?
6. Czym różni się fragmentacja zewnętrzna od wewnętrznej?

# Alokacja stronicowana

Metoda alokacji stronicowanej dzieli pamięć fizyczną na niewielkie bloki jednakowego rozmiaru, zwane **ramkami** (*frames*). Przestrzeń adresowa procesu jest podzielona na bloki o rozmiarze takim jak ramki, zwane **stronami** (*pages*).

Każdy proces posiada **tablicę stron** (*page table*) służącą do translacji adresów logicznych na fizyczne.

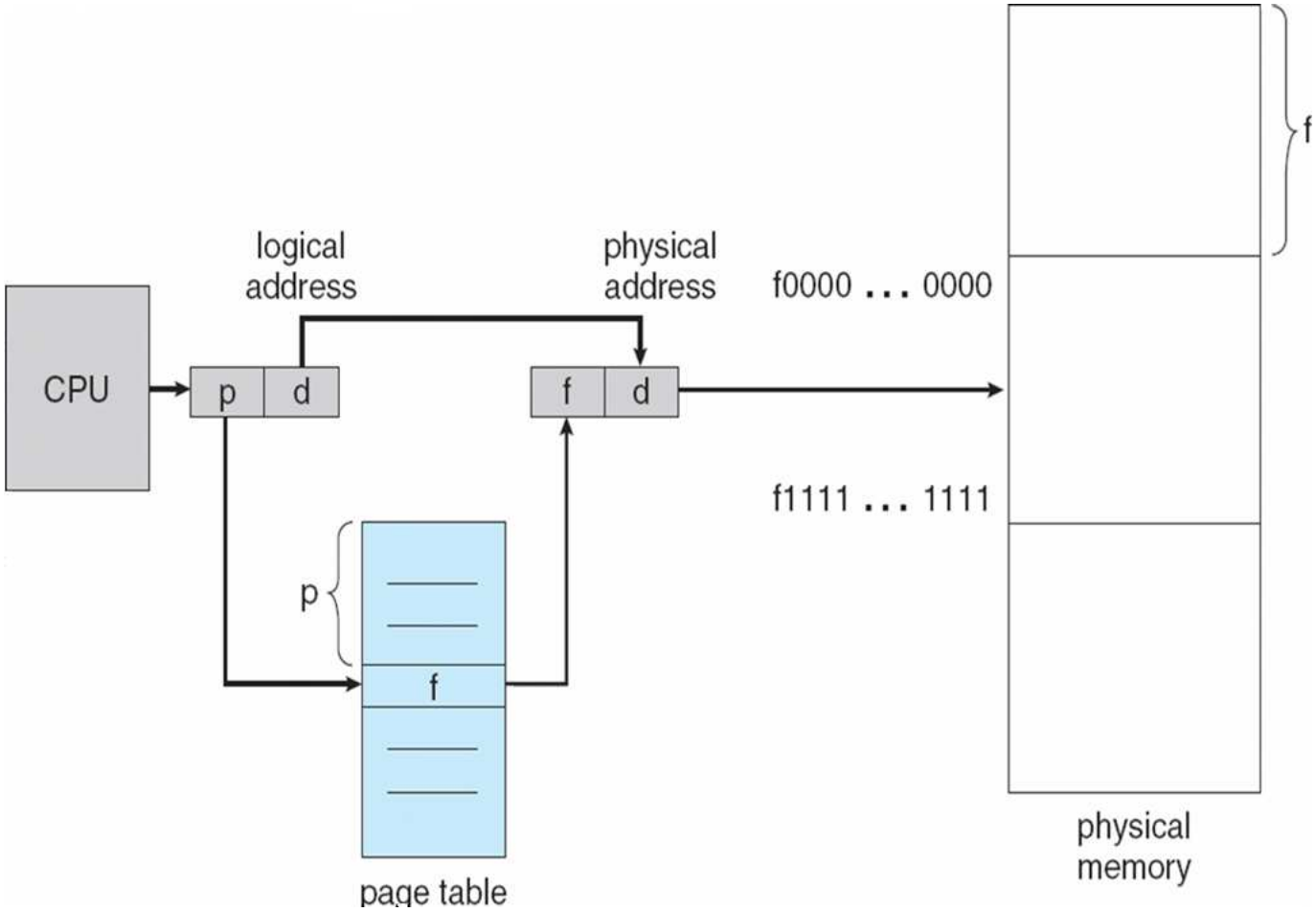
Tablica stron musi być oddzielna dla każdego procesu, ponieważ te same adresy logiczne powtarzają się dla wszystkich procesów, i muszą być odwzorowane do innych miejsc w pamięci fizycznej.



# Translacja adresu w alokacji stronicowanej

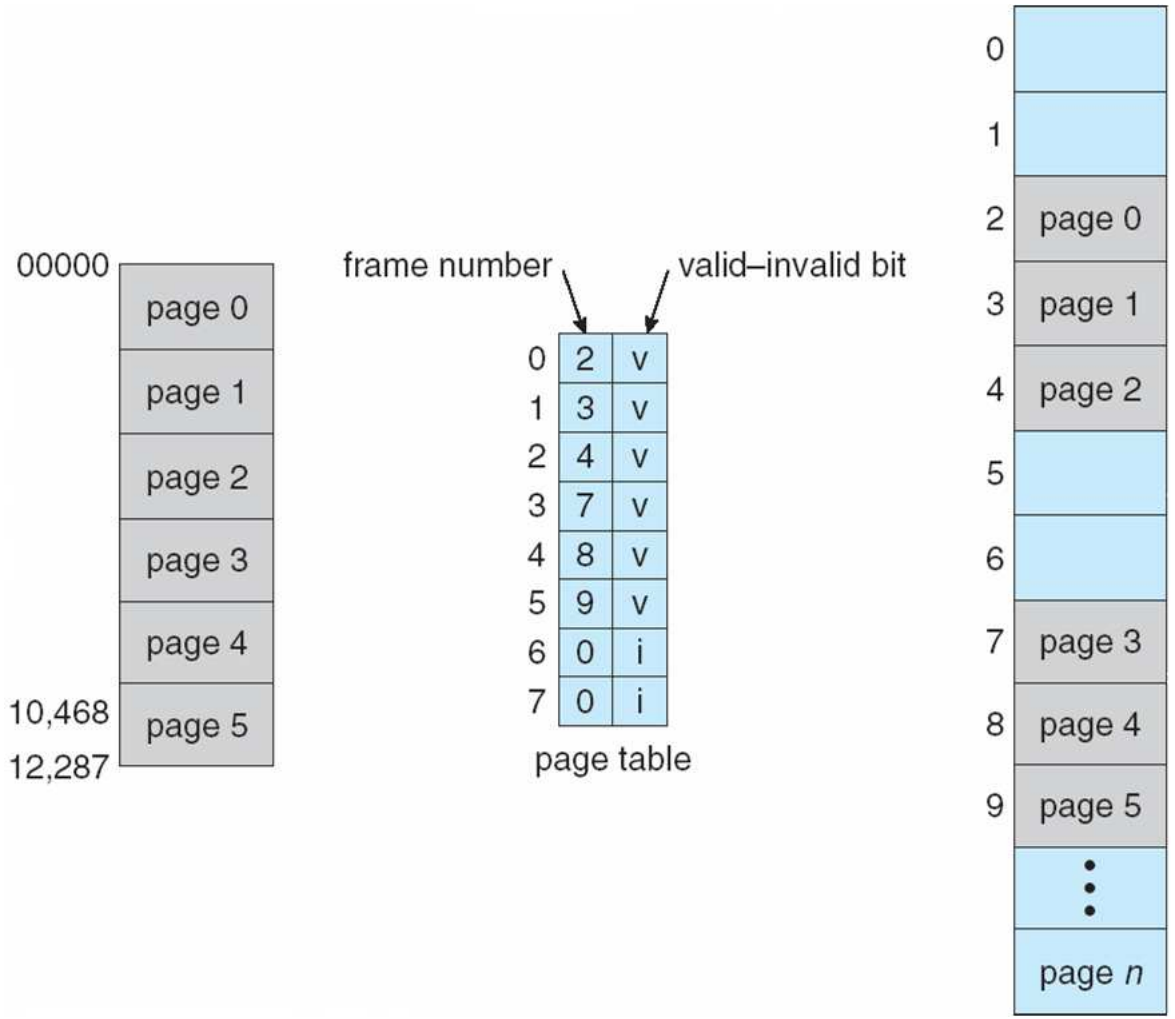
Adres logiczny składa się z dwóch części:

- numer strony (p) — służy jako indeks w tablicy stron,
- przesunięcie w stronie (d) — służy jako numer bajtu w stronie.



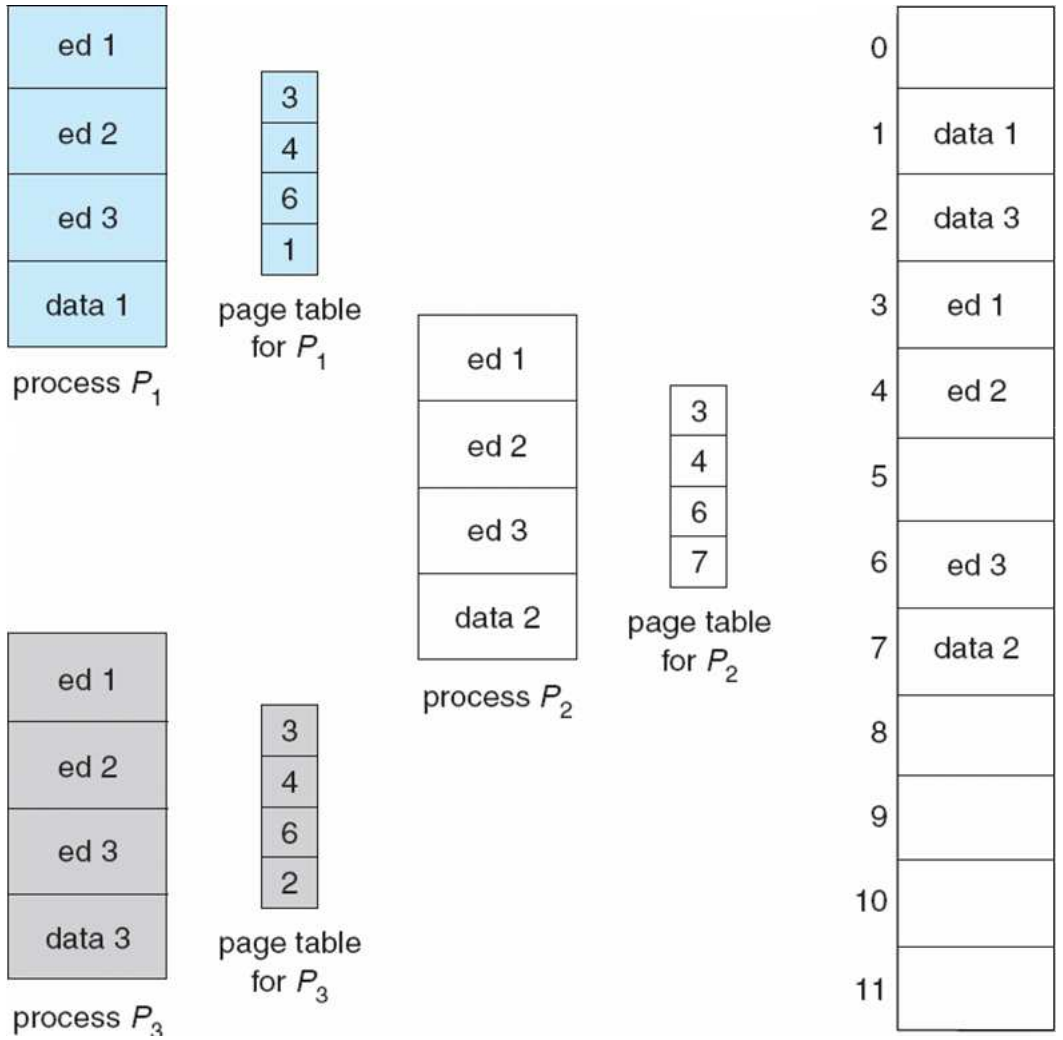
# Ochrona pamięci

Do zrealizowania ochrony pamięci, tzn. zapobiegania odwoływaniu się procesu do adresów pamięci, do których nie ma on praw dostępu, albo na których niedozwolona jest dana operacja, w tablicy stron istnieją dodatkowe informacje, takie jak: **bit ważności** (*valid bit*), albo bity praw dostępu: odczyt, zapis, tylko-wykonywanie.



# Współdzielenie stron przez różne procesy

Przykład realizacji alokacji stronicowanej ze współdzieleniem stron dla edytora, którego część pamięci zajmuje kod programu, wykorzystywany wyłącznie w trybie odczytu/wykonywania, z wydzielonym na oddzielnej stronie segmentem danych wykorzystywanym w trybie odczytu/zapisu:



# Własności systemów alokacji stronicowanej

System alokacji stronicowanej rozwiązuje problem dopasowania różnej wielkości segmentów alokacji pamięci do dostępnych obszarów w pamięci fizycznej. W ten sposób całkowicie unika się fragmentacji zewnętrznej.

Problem fragmentacji wewnętrznej jednak pozostaje. Aby go zmniejszyć, należałoby używać jak najmniejszych stron. To jednak powoduje wydłużanie tablic stron. Ponadto, wydajność transferu stron z i do pamięci jest lepsza dla stron o większym rozmiarze. Zatem rozmiar strony jest kompromisem w systemach operacyjnych — jest to kompromis pomiędzy zagadnieniem wydajności a fragmentacji wewnętrznej. (Na przykład, w systemie Solaris rozmiar strony może wynosić 8kB lub 4MB.)

Ponieważ każde odwołanie do pamięci przechodzi przez proces translacji adresu, system stronicowania musi być niezwykle szybki i wydajny. Jest on typowo realizowany przy wsparciu sprzętowym jednostki MMU. Wiele struktur, w tym tablica stron, musi być przechowywanych w rejestrach sprzętowych, a system operacyjny utrzymuje ich kopie. Operacje na tych strukturach typowo wydłużają czas przełączania kontekstu.

System operacyjny posiada również informacje o przydziale pamięci fizycznej (np. dostępnych wolnych ramkach) w **tablicy ramek**.

# Przechowywanie tablic stron

Tablice stron wszystkich procesów są przechowywane w pamięci RAM komputera co z jednej strony powoduje wydłużenie dostępu do pamięci (każdy dostęp wymaga najpierw pobrania wartości z tablicy stron w celu zdekodowania adresu, i dopiero wtedy pobrania rzeczywistej docelowej wartości danych), ale z drugiej strony wymaga przechowywania pewnej dodatkowej ilości danych.

Pierwszy problem jest rozwiązywany przez przechowywanie często używanych translacji adresów w specjalnej pamięci buforowej zwanej często TLB (*Translation Look-aside Buffer*).

Jednak drugi problem jest nietrywialny. Np. dla procesora 32-bitowego adresy są 32-bitowe, co przy rozmiarze ramki równym 4kB dzieli adres na 12-bitowy offset i 20-bitowy numer strony. Tablica stron procesu musi zatem mieć $2^{20}$ pozycji, co przy zaledwie czterech bajtach na pojedynczą pozycję (pozwalającą na zaadresowanie 4GB RAM) wymaga $2^{22}$ bajtów, czyli 4MB. Jednak tablica jest potrzebna w procesie translacji adresu, i musi być umieszczona w ciągłym obszarze pamięci. Jeśli w systemie uruchomionych zostanie np. 200 procesów, to wymaga to rzędu 1GB pamięci RAM. Chcąc wykorzystać alokację stronicowaną, w efekcie będziemy stronicować tablice stron!

# Przechowywanie tablic stron — adresy 64-bitowe

W przypadku procesorów 64-bitowych powyższe rozwiązanie problemu przechowywania tablic stron w pamięci już nie wystarczą, bo wymagałyby 7 poziomów stronicowania tablic stron. Logiczna przestrzeń adresowa takiej maszyny jest tak wielka, że przechowywanie pełnej tablicy stron jest nie do pomyślenia. Jest też zresztą niepotrzebne, bo procesy nie wykorzystują całej takiej przestrzeni adresowej.

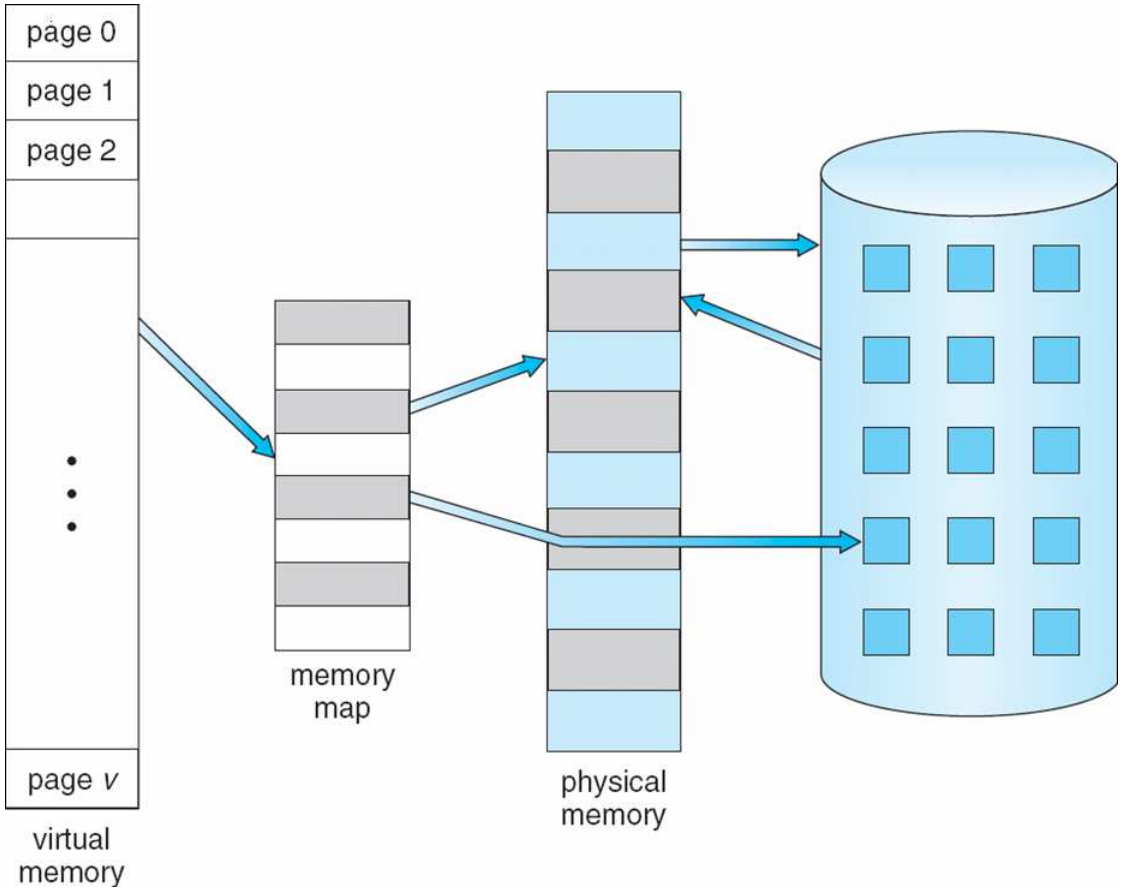
W takim przypadku można zastosować **odwróconą tablicę stron**, która odwzorowuje adresy ramek do adresów logicznych, zamiast na odwrót. Tablica stron zawierać wtedy będzie jedynie pozycje ramek wykorzystanych przez proces. Jednak by dokonać translacji adresu, trzeba taką tablicę przeszukiwać. Aby dokonać tego bardzo szybko, można zastosować strukturę taką jak tablica haszowa (**hash table**).

# Krótkie podsumowanie — pytania sprawdzające

1. Co to jest tablica stron?
2. Na czym polega proces translacji adresu w systemie alokacji stronicowanej?
3. Jak może być zrealizowana ochrona pamięci w systemie stronicowania?
4. Od czego zależy rozmiar strony pamięci?
5. Jaką korzyść daje realizacja sprzętowa systemu stronicowania?

# Pamięć wirtualna

Najczęściej wykorzystywanym w systemach operacyjnych schematem zarządzania pamięcią jest mechanizm **pamięci wirtualnej**. Polega on na całkowitym odseparowaniu pamięci logicznej użytkownika od pamięci fizycznej obsługiwanej przez system. Program użytkownika posługuje się wyłącznie liniową i adresowaną w sposób ciągły od zera pamięcią, natomiast system dzieli tę pamięć na małe fragmenty zwane stronami, i sam decyduje, które z nich są w danej chwili **rezydentne** w pamięci fizycznej. Oprócz tego wszystkie strony zapisane są na dysku w specjalnym **obszarze wymiany** (*swap space*).



# Zalety pamięci wirtualnej

W schemacie pamięci wirtualnej możliwe jest wykonywanie procesu, którego tylko niewielka część — w postaci wybranych stron pamięci — jest załadowana do pamięci fizycznej. Dzięki temu jest możliwe jednoczesne załadowanie wielu procesów do pamięci, i ich jednoczesne lub naprzemienne wykonywanie, nawet jeśli łączna objętość ich pamięci przekracza pojemność pamięci systemu.

Jest również możliwe wykonywanie programu, którego indywidualne zapotrzebowanie pamięci przekracza pojemność całej dostępnej pamięci fizycznej systemu.

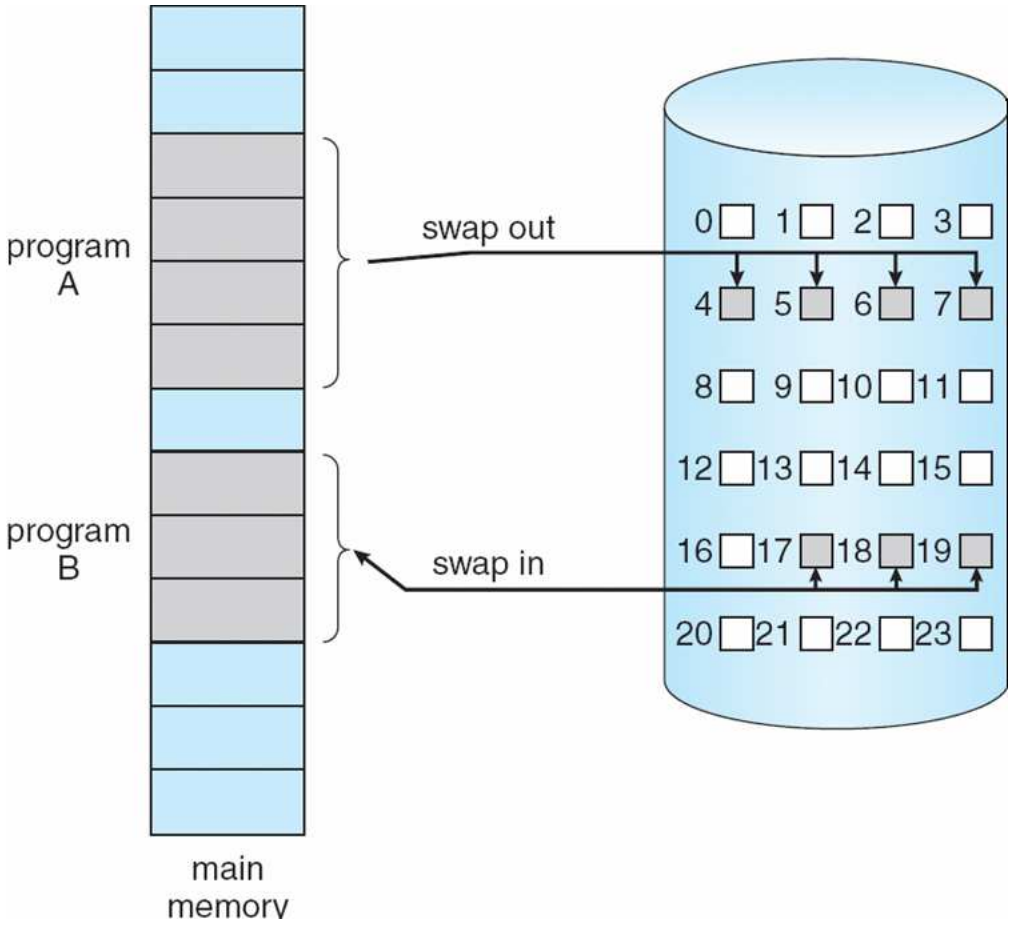
Ponadto, jest możliwe współdzielenie stron pamięci przez różne procesy, co ma zastosowanie np. przy korzystaniu z bibliotek współdzielonych.

W celu realizacji pamięci wirtualnej potrzebne są jednak odpowiednie mechanizmy, których część wymaga realizacji lub wsparcia sprzętowego:

- **stronicowanie**, czyli sprowadzanie potrzebnych stron z dysku do pamięci,
- **zastępowanie stron**, czyli usuwanie z pamięci chwilowo niepotrzebnych stron.

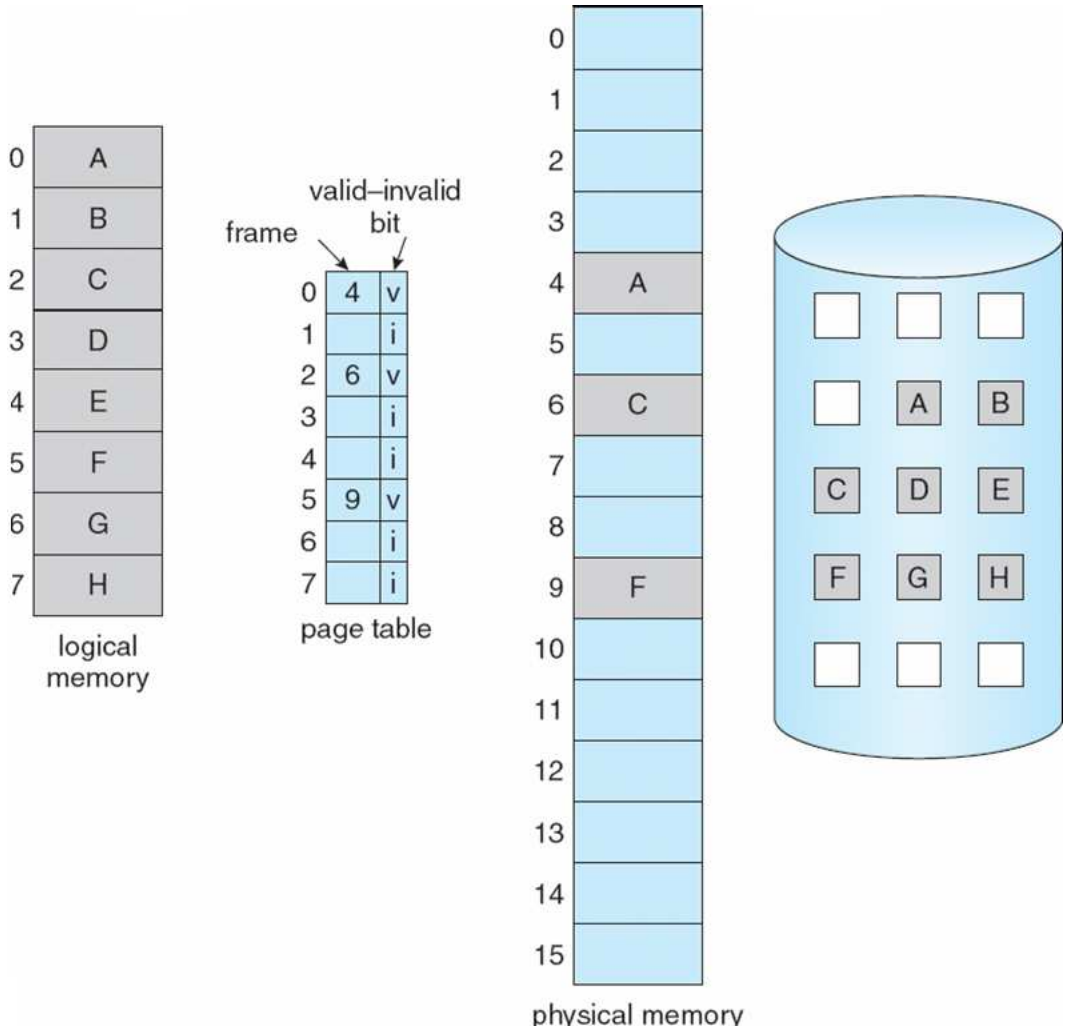
# Stronicowanie na żądanie

Najczęściej stosowanym mechanizmem implementacji pamięci wirtualnej jest **stronicowanie na żądanie** (*demand paging*). Polega on na sprowadzaniu do pamięci fizycznej tylko takich stron pamięci, do których odwołuje się dany proces. Strony niepotrzebne w danej chwili nie są automatycznie ładowane do pamięci.



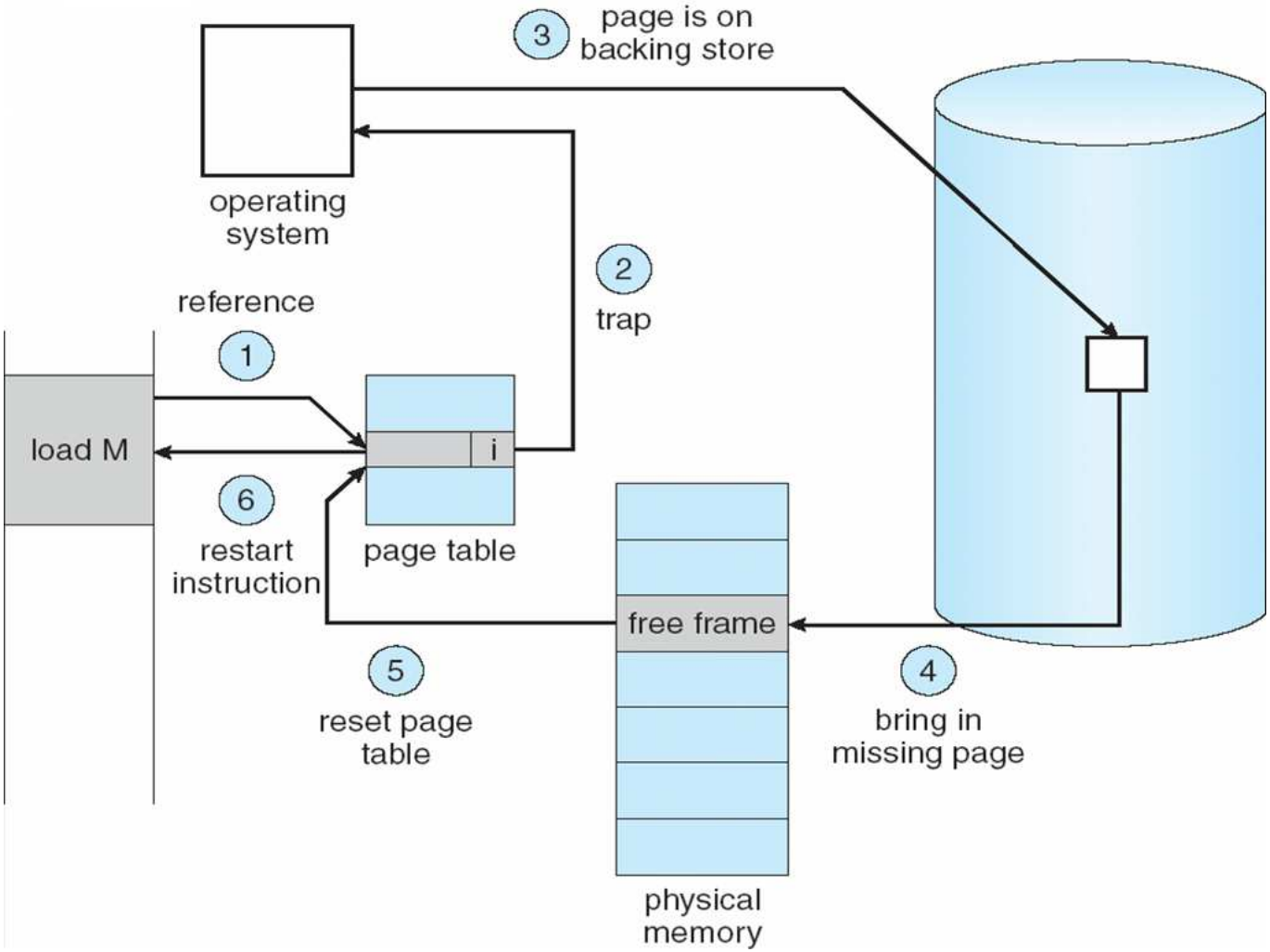
# Bit ważności

W czasie pracy systemu systematycznie przesyła on strony pamięci pomiędzy pamięcią fizyczną, a obszarem wymiany na dysku. Na skutek tego może się zdarzyć, że strona, która jakiś czas temu była w pamięci fizycznej została odesłana na dysk, i w chwili ponownej próby dostępu nie ma jej już w pamięci. Dlatego w tablicy stron potrzebne są bity ważności.



# Procedura obsługi błędu strony

Normalny algorytm translacji adresu stronicowanego musi być uzupełniony o obsługę wyjątku ② **błędu strony** w pamięci (*page fault trap*):



# Wydajność stronicowania na żądanie

Etap ściągania z dysku strony pamięci jest najdłuższym elementem procedury obsługi błędu strony. Dramatycznie wydłuża on wykonanie instrukcji, która go wywołała (tysiące do dziesiątków tysięcy razy przy wykorzystaniu magnetycznych dysków obrotowych).

Mogłoby nasuwać się pytanie, czy procedura stronicowania na żądanie jest skuteczna? Co byłoby w przypadku, gdyby wykonanie jednej instrukcji skutkowało wyjątkiem błędu strony, a po jej sprowadzeniu do pamięci i wznowieniu programu, kolejne instrukcje generowały dalsze błędy strony?

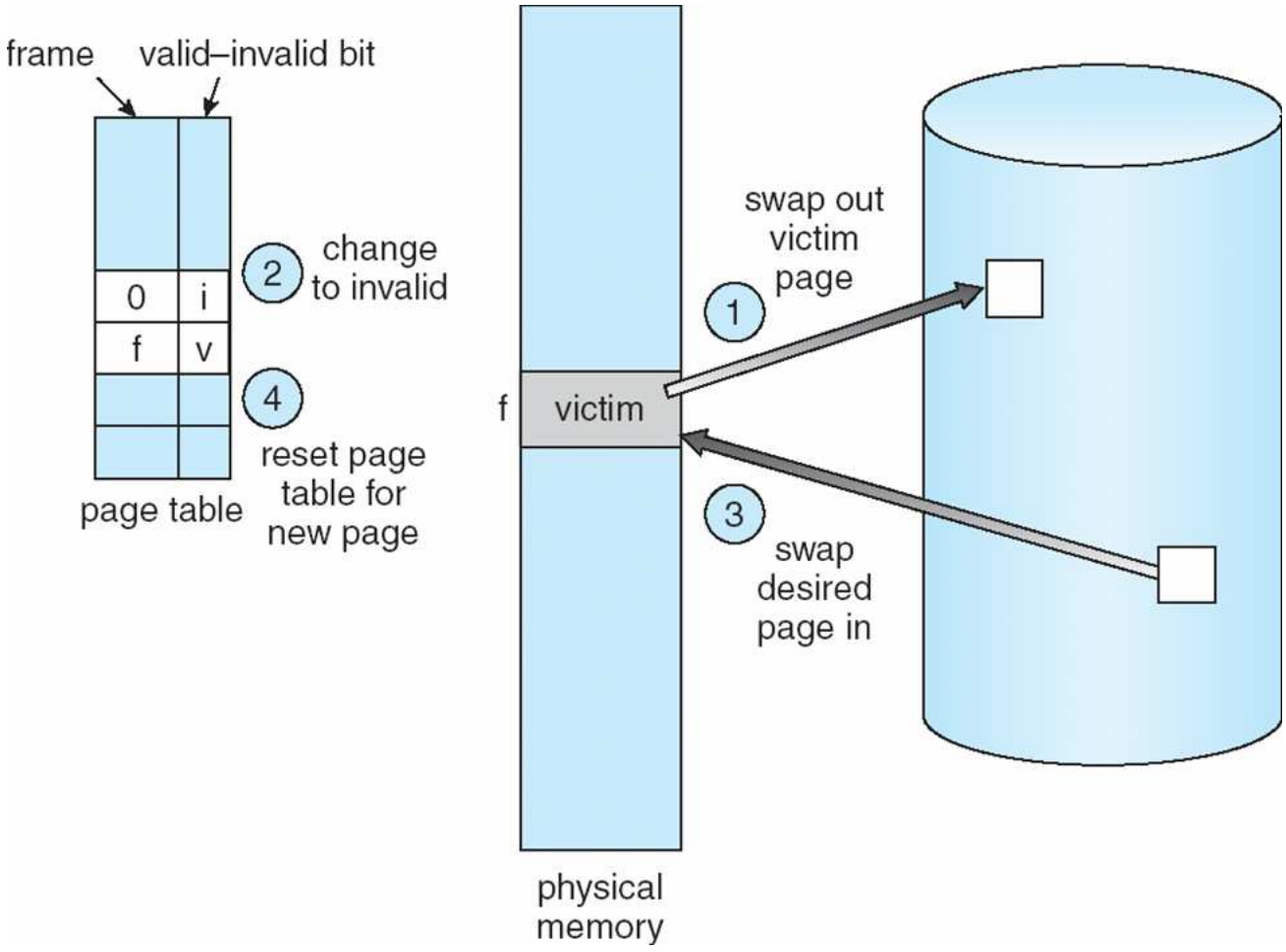
W oczywisty sposób, system działałby wtedy bardzo niewydajnie.

Praktyczne analizy wykazują, że takie sytuacje zdarzają się rzadko, natomiast najczęściej dominuje **zasada lokalności**, polegająca na tym, że jeśli dana strona została przeniesiona do pamięci, to często wiele kolejnych instrukcji programu z niej korzysta, np. pętle.

# Zastępowanie stron

Procedura obsługi wyjątku błędu strony w pamięci działa poprawnie, gdy w pamięci jest wolna strona, w miejsce której można skopiować potrzebną stronę z obszaru wymiany. Co jednak zrobić w przypadku, gdy wszystkie strony w pamięci są już zajęte? W takim przypadku system pamięci wirtualnej musi wytypować stronę zwaną **ofiarą** (*victim*), usunąć ją z pamięci, i wykorzystać powstałe wolne miejsce w pamięci RAM.

Wiąże się to z dodatkową operacją zapisu zawartości tej strony na dysku, i aktualizacją odpowiednich tablic stron.



# Zastępowanie stron — ofiary brudne

Procedura wyboru ofiary i usuwania jej z pamięci nazywana jest **zastępowaniem stron** (*page replacement*).

Zauważmy najpierw, że wytypowanej strony ofiary nie zawsze trzeba przepisywać z powrotem do obszaru wymiany na dysku. Jeśli nie została ona zmieniona przez program, to można po prostu zapomnieć jej zawartość i nadpisać ją w pamięci.

W tym celu stosuje się dodatkowy bit w tablicy stron, zwany **bitem brudnym** (*dirty bit*). Bity te są początkowo zerowane, i ustawiane przy zapisie wykonywanym do strony w pamięci. Jeśli przy zastępowaniu strony bit brudny jest wyzerowany, to ofiarę można bez konsekwencji po prostu nadpisać w pamięci.

Natomiast ofiarę z ustawionym bitem brudnym, która musi być kopiowana na dysk, nazywa się **ofiarą brudną** (*dirty victim*).

Z tego punktu widzenia, w oczywisty sposób, najlepiej byłoby wybierać, w miarę możliwości, ofiary czyste!! Jednak to nie czystość stanowi właściwe kryterium wyboru ofiar.

# Zastępowanie stron — algorytm wyboru ofiary

Wybór ofiary jest kluczowym elementem procesu zastępowania stron, ponieważ niewłaściwe wybieranie ofiar może drastycznie ograniczyć efektywność funkcjonowania całego systemu pamięci wirtualnej, wynikającą z zasady lokalności.

Podstawowe algorytmy wyboru ofiary:

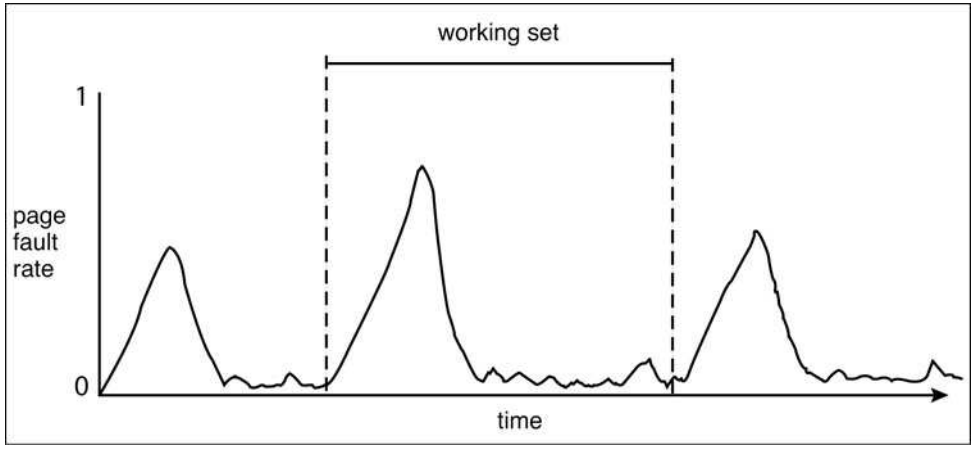
**algorytm FIFO** — zdumiewająco nieskuteczny,

**algorytm OPT** (optymalny, nierealizowalny w praktyce) — wymaga analizy zachowania się procesów w przyszłości, i wybiera jako ofiarę tę stronę, która najdłużej nie będzie potrzebna,

**algorytm LRU** (*least recently used*) — jest jakby odbiciem algorytmu OPT w przeszłości; jest bardzo skuteczny dzięki zasadzie, że procesy na ogół realizują przez pewien czas podobne operacje; a więc w najbliższej przyszłości będą robiły to, co robiły w najbliższej przeszłości; algorytm LRU jest kosztowny w realizacji (sprzętowej), stąd dawniej był wykorzystywany rzadziej, obecnie częściej; stosowane są również różne przybliżenia pełnego LRU,

**algorytm LFU** (*least frequently used*) — wydaje się atrakcyjnym pomysłem, lecz w praktyce przybliża OPT gorzej niż LRU lub jego przybliżenia.

# Zbiory robocze



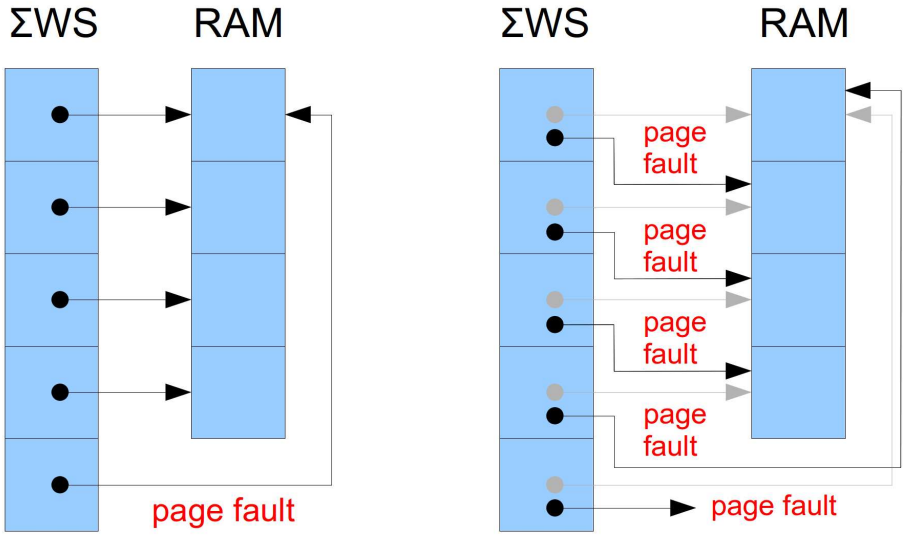

Można zadać sobie pytanie, ile pamięci fizycznej naprawdę potrzebuje system, aby pracować wydajnie. Może miałoby sens utrzymywanie puli wolnych stron?

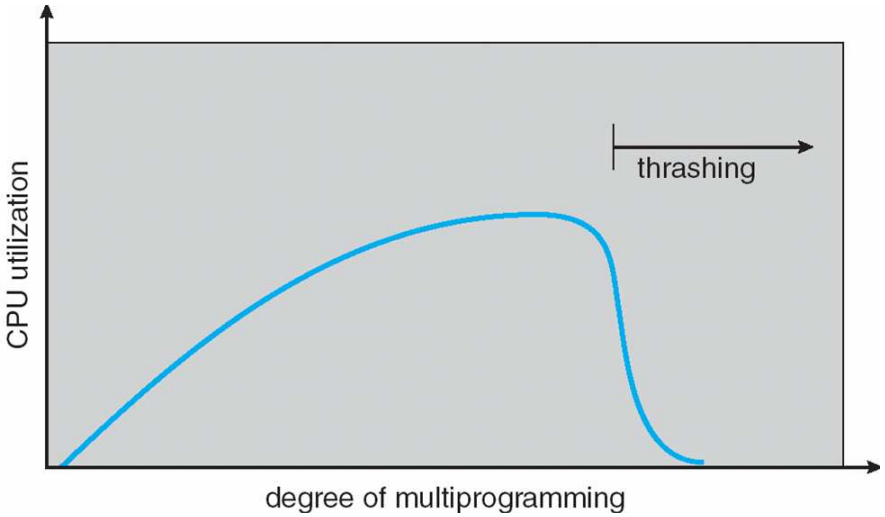
Zestaw stron, w ramach którego proces pracuje stabilnie przez pewien czas, bez generowania dużej liczby wyjątków błędu strony, nazywamy jego **zbiorem roboczym** (*working set*). Co jakiś czas, każdy proces typowo przechodzi do innej fazy obliczeń, co jest związane z intensywnym stronicowaniem, zanim ustabilizuje się jego nowy zbiór roboczy, i aktywność stronicowania spada.

Aby system pamięci wirtualnej pracował poprawnie, suma zbiorów roboczych wszystkich procesów w kolejce gotowych, plus pamięć wykorzystywana przez system, nie może przekraczać całej dostępnej pamięci fizycznej: $\sum WS + OS \leq RAM$

# Szamotanie

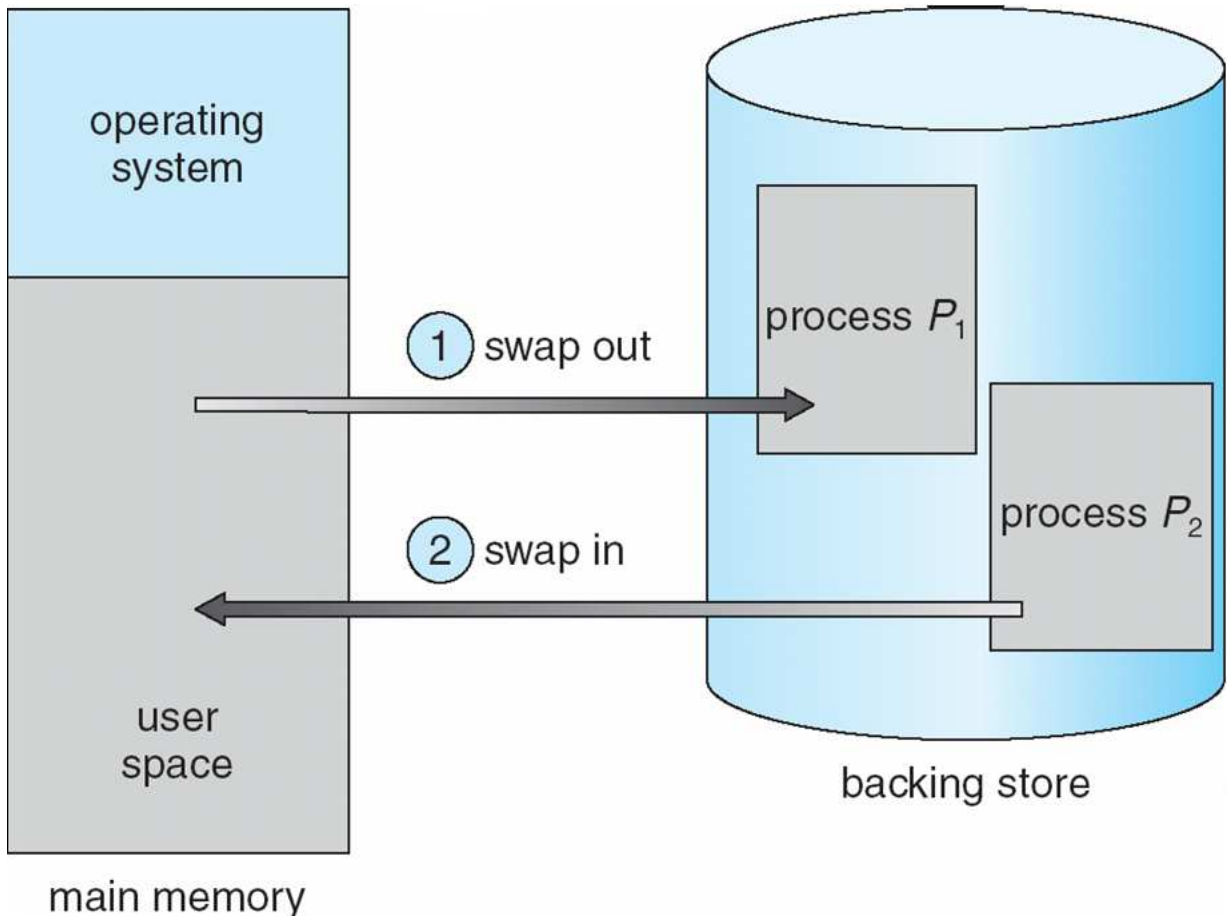
Jeśli suma zbiorów roboczych wszystkich procesów, wraz z pamięcią wykorzystywaną przez system, przekroczy choć minimalnie dostępną pamięć fizyczną komputera, następuje katastrofa. Algorytm zastępowania stron zaczyna wtedy działać na niekorzyść systemu, powodując wielką liczbę wyjątków braku strony. Zjawisko to jest nazywane **szamotaniem** (*thrashing*), i drastycznie zmniejsza wydajność pracy systemu.




Ponieważ procedura obsługi wyjątku braku strony trwa wielokrotnie dłużej niż zwykły dostęp do pamięci (wiele rzędów wielkości dłużej), użytkownicy odczuwają to jakby system przestał w ogóle wykonywać ich procesy.

# Wymiatanie




**Wymiatanie** (*swapping*) jest procedurą chwilowego usuwania całych procesów z pamięci, stosowaną jako obrona przed szamotaniem. Jeśli po usunięciu jednego lub więcej procesów, suma zbiorów roboczych pozostałych mieści się w pamięci fizycznej, system wznawia normalną pracę. Po jakimś czasie wymiecione procesy przywracane są do pamięci, a w razie potrzeby system wymiata na dysk inne.

# Pamięć buforowa cache

Jak już wiemy, pamięć operacyjna realizowana jest w technologii wielopiętrowej piramidy, w której pomiędzy pamięcią operacyjną (RAM) a procesorem stosowane są mniejsze ilości znacznie szybszej pamięci buforowej **cache**, której funkcjonowanie uzasadnione jest również zasadą lokalności — często odwołania do strony pamięci następują seriami, i każde kolejne odwołanie pobierane jest już z pamięci buforowej.

Należy pamiętać, że pamięć buforowa jest zrealizowana sprzętowo, poniżej mechanizmów pamięci wirtualnej, i nie ma ona świadomości istnienia procesów, i ich wirtualnych przestrzeni adresowych. Dlatego adresowanie na poziomie pamięci buforowej odbywa się zawsze w kategoriach adresów fizycznych.

# Krótkie podsumowanie — pytania sprawdzające

1. Czym różni się system pamięci wirtualnej od zwykłego systemu stronicowania?
2. Jaki wyjątek musi obsłużyć procedura translacji adresu w systemie pamięci wirtualnej, i jak ta obsługa przebiega?
3. Co to jest bit ważności w tablicy stron procesu, i jaka jest jego rola w systemie pamięci wirtualnej?
4. Co to jest ofiara (*victim*) i jakie operacje są na niej wykonywane w systemie pamięci wirtualnej?
5. Wymień podstawowe algorytmy zastępowania stron pamięci wirtualnej.
6. Co to są zbiory robocze?
7. Na czym polega zjawisko szamotania i jak można mu zapobiegać?
8. Czy do pamięci cache można odwoływać się adresem wirtualnym? A czy można fizycznym? Dlaczego?