# Systemy operacyjne

## Wykład #6

## Zarządzanie pamięcią

- Podstawy
- Wymiana (swapping)
- Przydział ciągły pamięci
- Stronicowanie

wersja:1.10.2021

# Zarządzanie pamięcią podstawy

- pamięć **operacyjna** (główna) (*main memory, core*) → tablica słów lub bajtów, indeksowana adresem
- aby program mógł być wykonywany, musi zostać umieszczony w procesie i wprowadzony do pamięci operacyjnej
- kolejka wejściowa (*input queue*) → zbiór procesów czekających na dysku na wprowadzenie do pamięci w celu wykonania
- jeden z procesów zostaje wybrany i załadowany do pamięci
- podczas wykonywania pobiera rozkazy i dane z pamięci, a po zakończeniu zwalnia zajmowaną pamięć

# Zarządzanie pamięcią podstawy

- większość systemów pozwala procesowi użytkownika przebywać w dowolnej części pamięci fizycznej → wpływa to na zakres adresów dostępnych dla procesu
- program użytkownika przechodzi kilka faz zanim zostanie wykonany → reprezentacja adresów może ulegać zmianie
- program źródłowy → adresy wyrażone w sposób symboliczny → kompilator wiąże je z adresami względnymi; linker (konsolidator) może powiązać adresy względne z adresami bezwzględnymi
- sposób i moment wykonania wiązania decyduje o wydajności i elastyczności całego systemu operacyjnego

# Wiązanie adresów

Wiązanie rozkazów i danych z adresami pamięci może być dokonane w dowolnym z trzech kroków:

- **kompilacja** → jeżeli a priori znane jest miejsce, w którym proces będzie przebywał w pamięci, to można wygenerować kod bezwzględny (*absolute code*); zmiana położenia kodu w pamięci wymaga jego rekompilacji

- **ładowanie** → jeżeli przyszłe położenie procesu w pamięci nie jest znane podczas kompilacji, to kompilator musi generować kod przemieszczalny (*relocatable code*); wiązanie adresów następuje w czasie ładowania kodu

- **wykonanie** → jeżeli proces podczas wykonania może być przemieszczany w pamięci, to wiązanie adresów musi być opóźnione do czasu wykonania; wymaga to specjalnego sprzętu; stosowane w większości współczesnych architektur

# Przestrzeń adresowa

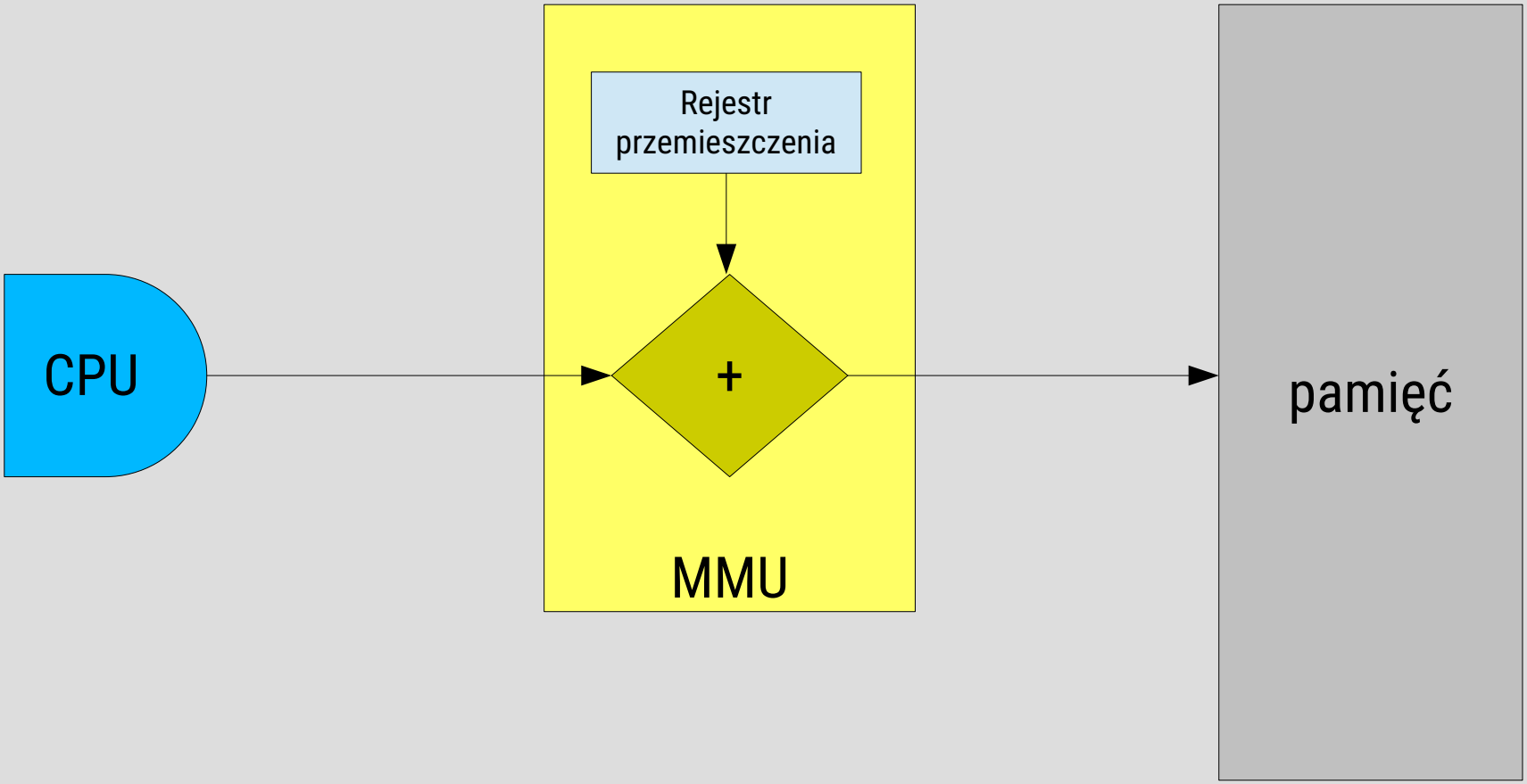
- adresy **logiczne** (wirtualne) – adresy generowane przez CPU
- adresy **fizyczne** – adresy widziane przez jednostkę pamięci
- adresy logiczne i fizyczne są:
  - takie same – w schematach wiązania adresów podczas kompilacji i ładowania;
  - różne – w schematach wiązania adresów podczas wykonywania.

- zbiór wszystkich adresów logicznych generowanych przez program → logiczna przestrzeń adresowa
- zbiór odpowiadających im adresów fizycznych → fizyczna przestrzeń adresowa
- współczesne architektury sprzętowe umożliwiają konstruowanie obu przestrzeni niemal niezależnie
- w szczególności obie przestrzenie mogą różnić się rozmiarem

# Przestrzeń adresowa

- odwzorowywanie adresów logicznych na fizyczne jest dokonywane przez jednostkę zarządzania pamięcią (*memory management unit* – **MMU**); jest to struktura sprzętowa
- jeden z możliwych sposobów działania MMU: do każdego adresu generowanego przez proces użytkownika w chwili odwołania się do pamięci dodawana jest wartość rejestru przemieszczenia (*relocation register*)
- program użytkownika działa **tylko** na adresach logicznych, **nigdy** nie ma do czynienia z rzeczywistymi adresami fizycznymi

# Przestrzeń adresowa



# Konsolidacja

**konsolidacja statyczna** – (*static linking*) systemowe biblioteki języków programowania są traktowane jak każdy inny moduł wynikowy i dołączane przez program ładujący do binarnego obrazu programu

**Wada**:
marnotrawstwo pamięci operacyjnej oraz przestrzeni dyskowej (dublowanie kodu)

**Zaleta:**
gotowy kod można przenosić między systemami bez konieczności instalacji bibliotek

# Konsolidacja

**konsolidacja dynamiczna** (*dynamic linking*) – konsolidacja jest opóźniana do czasu wykonania programu

- w miejscu odwołania do kodu bibliotecznego w obrazie binarnym procesu znajduje się tzw. *stub* (namiastka procedury, dosł. *kikut*) → mały fragment kodu wskazujący, jak odpowiedni podprogram odnaleźć w pamięci lub załadować z dysku

- *stub* wprowadza na swoje miejsce adres potrzebnego podprogramu i powoduje jego wykonanie

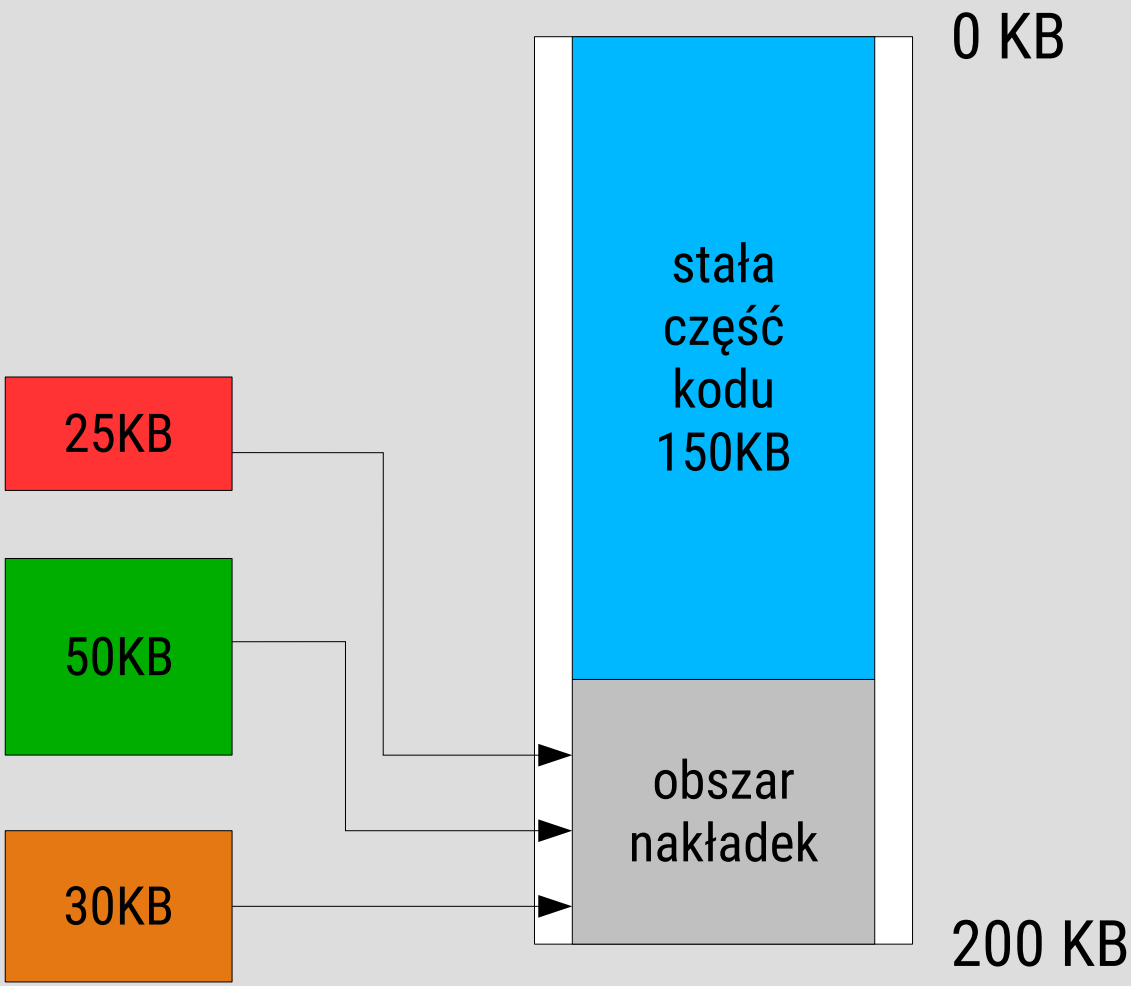
# Konsolidacja

- wymaga wsparcia ze strony sytemu operacyjnego → sprawdzenie, czy podprogram biblioteczny jest w obszarach pamięci innych procesów oraz umożliwienie, by wiele procesów miało dostęp do tych samych adresów pamięci (jeśli procesy w pamięci są chronione przed sobą nawzajem)
- lepsze wykorzystanie pamięci operacyjnej oraz przestrzeni dyskowej
- szczególnie użyteczna dla bibliotek, np. aktualizacja biblioteki nie wymaga ponownej konsolidacji korzystających z niej programów
- biblioteki dzielone (*shared libraries*) →
  - pliki *.so w Linuksach
  - *.dll w Windows

# Nakładki (*overlays*)

- przechowywanie w pamięci tylko tych rozkazów i danych, które są stale potrzebne
- inne rozkazy i dane są wprowadzane w miarę zapotrzebowania na miejsca zajmowane przez rozkazy/dane już zbyteczne
- daje możliwość uruchamiania procesu o rozmiarze większym od przydzielonego mu obszaru pamięci
- nie wymaga specjalnego wsparcia ze strony systemu operacyjnego
- może być w całości implementowane przez użytkownika
- projektowanie i programowanie nakładek jest skomplikowane i obecnie rzadko stosowane

# Nakładki (*overlays*)



# Wymiana (*swapping*)

- proces może być tymczasowo **odesłany** z pamięci operacyjnej do pamięci pomocniczej, a następnie sprowadzony z powrotem w celu kontynuacji wykonywania
- do wymiany potrzebna jest pamięć pomocnicza (*backing store*) → na ogół **szybki** dysk
- pamięć pomocnicza musi być wystarczająco pojemna, aby pomieścić kopie obrazów pamięci wszystkich procesów
- musi umożliwiać bezpośredni dostęp do tych obrazów pamięci

# Wymiana (*swapping*)

- **wytaczanie** i **wtaczanie** (*roll-out*, *roll-in*) – wariant wymiany używany w algorytmach planowania priorytetowego, w którym proces o niższym priorytecie jest odsyłany z pamięci, by proces o wyższym priorytecie mógł zostać załadowany i wykonany

- najbardziej znaczącym składnikiem czasu wymiany jest czas przesyłania, który jest proporcjonalny do rozmiaru wymienianej pamięci

- zmodyfikowane wersje wymiany stosowane są w większości współczesnych systemów operacyjnych

# Ciągły przydział pamięci

alokacja pamięci:

- **dziura** (hole) → blok dostępnej (nieprzydzielonej) pamięci
- dziury mogą być rozsiane po całej pamięci
- pojawienie się nowego procesu → poszukiwanie odpowiednio obszernej dziury
- brak odpowiedniej dziury → zatrzymanie procesu
- system operacyjny musi:
    - prowadzić ewidencję przydzielonych bloków pamięci
    - prowadzić ewidencję dziur

# Alokacja pamięci

Sposoby wyboru dziury dla procesu:

- **pierwsze dopasowanie** (*first-fit*) → przydziela się pierwszą dziurę o wystarczającej wielkości; szuka się od początku wykazu dziur aż do pierwszego trafienia albo od miejsca zakończenia ostatniego szukania
- **najlepsze dopasowanie** (*best-fit*) → przydziela się najmniejszą z dostatecznie dużych dziur; zapewnia najmniejsze pozostałości, ale wymaga przeszukiwania całej ewidencji
- **najgorsze dopasowanie** (*worst-fit*) → przydziela się największą dziurę; wymaga przeszukania całej ewidencji, pozostawia po przydziale największą dziurę (czasami może być bardziej przydatna niż wiele rozproszonych małych dziur)

# Alokacja pamięci

Symulacje pokazują, że *first-fit* i *best-fit* są **lepsze** od *worst-fit* pod względem zużycia czasu i pamięci (**najszybszy** jest *first-fit*)

# Fragmentacja pamięci

- **fragmentacja zewnętrzna** (*external fragmentation*) → suma wolnych obszarów w pamięci wystarcza na spełnienie zamówienia, ale nie tworzą one spójnego obszaru

- **reguła 50 procent** – w strategii pierwszego dopasowania straty z powodu fragmentacji wynoszą zwykle ok. 50% tego, co już zostało przydzielone

# Fragmentacja pamięci

- **fragmentacja wewnętrzna** (*internal fragmentation*) → przydzielona pamięć może być **nieco** większa niż zamówiona (bo nie opłaca się trzymać informacji o bardzo małych dziurach)
- naddatek znajduje się wewnątrz przydzielonego obszaru, ale nie jest wykorzystywany

# Fragmentacja pamięci

- fragmentację zewnętrzną można redukować przez upakowanie → przetasowanie pamięci tak, aby cała wolna pamięć była w jednym bloku (jedna dziura)

- możliwe tylko przy dynamicznym wiązaniu adresów wykonywanym podczas działania procesu

- **wady:**
  - algorytm upakowania może być kosztowny
  - wybranie optymalnej strategii upakowywania jest trudne
  - *swapping* → dodatkowe trudności: problem przywrócenia procesu o statycznie ustalanych adresach.

# Stronicowanie

- inne rozwiązanie problemu fragmentacji zewnętrznej: dopuszczenie nieciągłości logicznej przestrzeni adresowej → przydzielanie procesowi dowolnych dostępnych miejsc pamięci fizycznej, ale określonymi porcjami
- pamięć fizyczna podzielona jest na bloki o stałej długości zwane **ramkami** (*frames*); rozmiar ramki jest potęgą 2, zwykle między 512B a 16MB
- pamięć logiczna podzielona jest na bloki takiego samego rozmiaru zwane **stronami** (*pages*)
- system utrzymuje informację o wolnych i przydzielonych ramkach w tablicy ramek (pamięta się też do jakiego procesu/procesów ramka jest przydzielona)

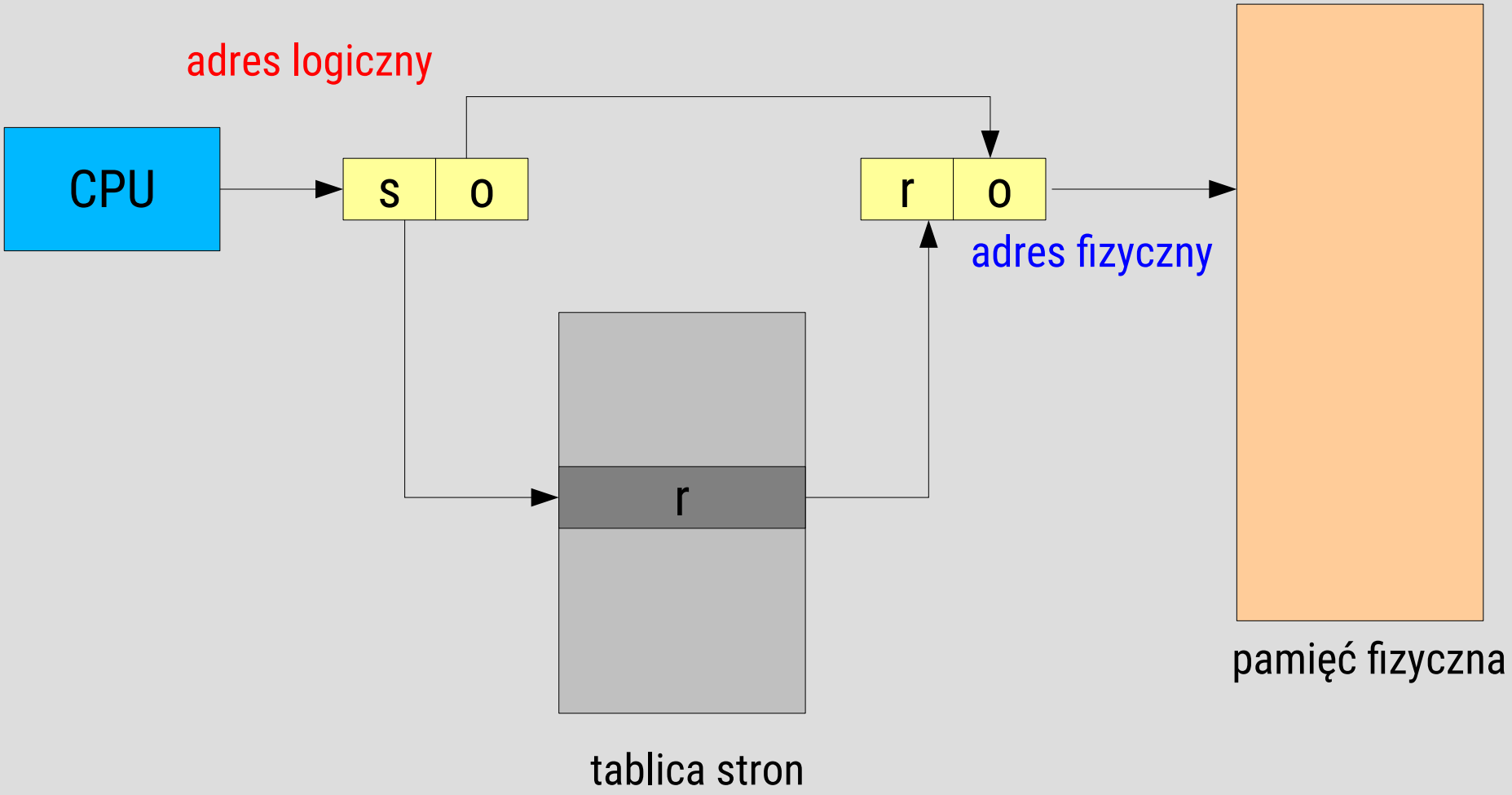
# Stronicowanie

- strony uruchamianego procesu przebywające w pamięci pomocniczej są wprowadzane w wolne ramki pamięci operacyjnej → program o rozmiarze *n* stron potrzebuje *n* ramek
- mogą być one rozmieszczone w dowolnych miejscach pamięci fizycznej
- eliminacja fragmentacji zewnętrznej
- nasilenie fragmentacji wewnętrznej

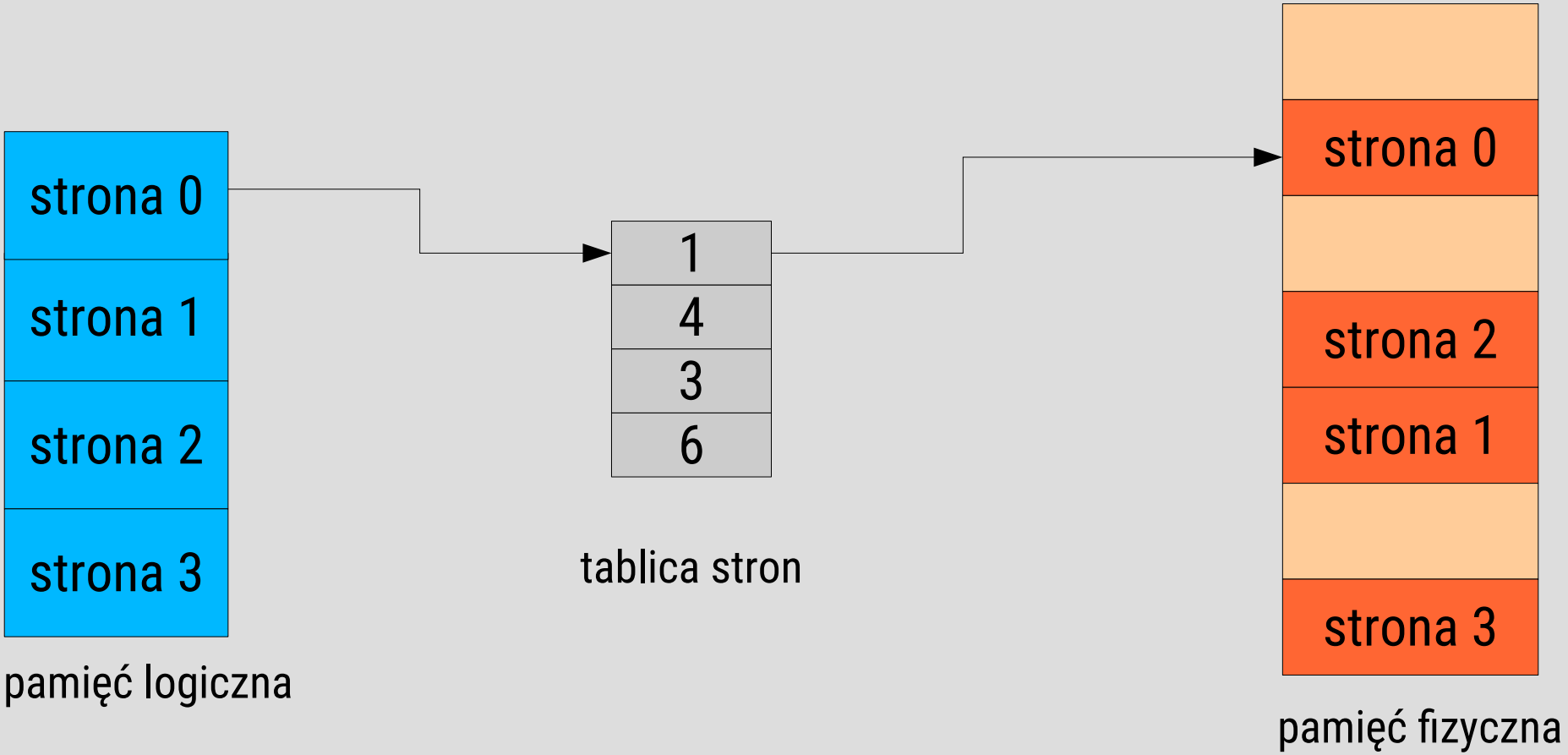
# Stronicowanie

- **adres logiczny** generowany przez procesor:
  - numer strony (*page number*) → indeks w tablicy stron; tablica stron zawiera adresy bazowe wszystkich stron w pamięci operacyjnej
  - odległość na stronie o (*page offset*) – w połączeniu z adresem bazowym daje adres fizyczny pamięci, który jest wysyłany do jednostki pamięci

# Stronicowanie



# Stronicowanie



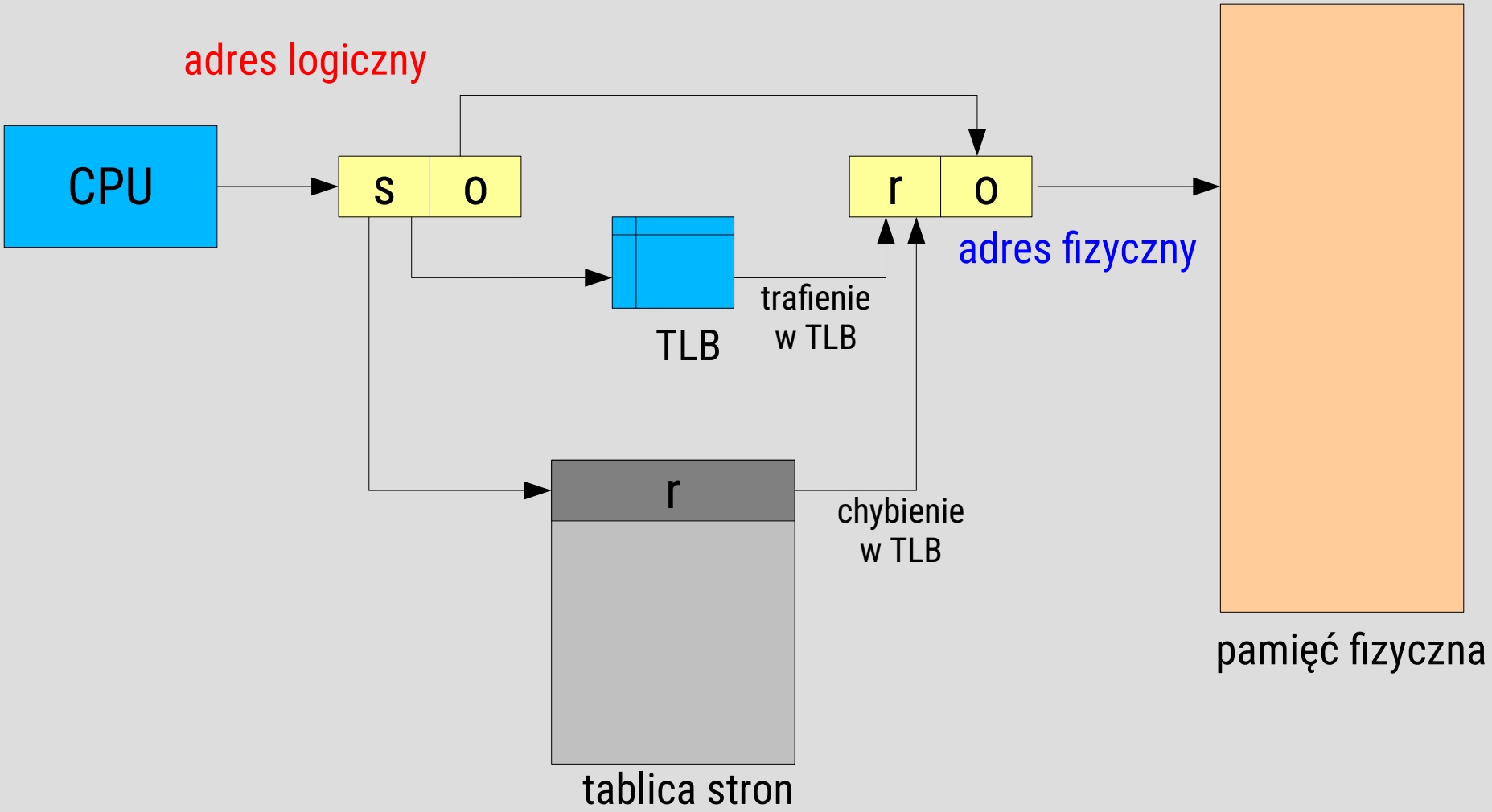
# Tablica stron

- **tablica stron** → przechowywana jest w pamięci operacyjnej (jedynie bardzo małe tablice można przechowywać w rejestrach)
- **rejestr bazowy tablicy stron** (*page-table base register* – PTBR) → wskazuje położenie tablicy stron
- **rejestr długości tablicy stron** (*page-table length register* – PTLR) → podaje rozmiar tablicy stron

# Tablica stron

- każdy dostęp dodanych/rozkazów wymaga dwóch **dodatkowych** kontaktów z pamięcią → trzykrotne spowolnienie dostępu
- standardowe rozwiązanie → użycie szybkiej (drogiej) pamięci podręcznej → bufor translacji adresów stron
- (*translation look-aside buffer* – TLB)
- każdy rejestr składa się z:
  - klucza (znacznika)
  - wartości
- porównywanie danego obiektu (numeru strony) z kluczami odbywa się równocześnie dla wszystkich kluczy
- jeżeli obiekt zostanie znaleziony, to wynikiem jest numer ramki;
- jeżeli nie, to trzeba odwołać się do tablicy stron w pamięci

# Stronicowanie z TLB



# Ochrona pamięci

- każdej ramce przypisuje się bity ochrony, które określają czy strona jest dostępna do czytania i pisania, czytania czy wykonania
- dodatkowo do każdej pozycji w tablicy stron dołączany jest bit poprawności (*valid-invalid bit*) mówiący, czy strona znajduje się w logicznej przestrzeni adresowej procesu (dozwolona) czy nie
- niektóre systemy korzystają ze sprzętowego rejestru długości tablicy stron (PTLR) – używa się go w celu sprawdzenia, czy dany adres należy do przedziału dozwolonego dla procesu