Podstawy systemów operacyjnych

Wykład 3

Klasyczne problemy synchronizacji procesów

Problem czytelników i pisarzy

- Obiekt danych podlega współbieżnemu dzieleniu między kilka współbieżnych procesów.
- Rozróżniamy 2 typy procesów:
  - zainteresowane czytaniem danych – **czytelnicy;**
  - zainteresowane aktualizacją danych – **pisarze**;
- Jednoczesny dostęp do obiektu dwóch procesów czytających nie spowoduje problemu, natomiast jednoczesny dostęp pisarza i któregokolwiek innego procesu mógłby spowodować chaos.
- Aby uniknąć problemów należy zagwarantować **wyłączność procesów pisarzy** do obiektów.

Odmiany

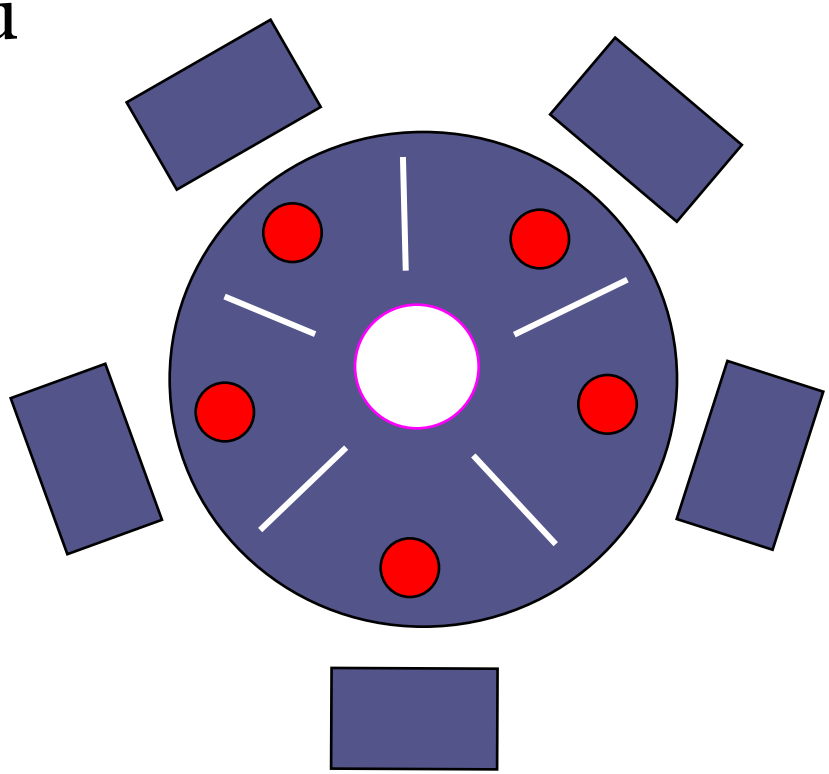
Istnieje kilka odmian z zastosowaniem priorytetów:

1. Zakłada się, że żaden czytelnik nie powinien czekać, chyba, że właśnie pisarz ma dostęp do obiektu.
2. Zakłada się, że pisarz otrzymuje dostęp do obiektu tak wcześnie jak to tylko jest możliwe. Czyli jeśli pisarz czeka na dostęp do obiektu to żaden czytelnik nie rozpocznie czytania

- Rozwiązanie każdego z powyższych problemów może powodować głodzenie pisarzy (1) lub czytelników(2).

Problem obiadujących filozofów

- 5 filozofów spędzających czas na myśleniu i jedzeniu



Problem obiadujących filozofów

- Problem ten uznawany jest za klasyczne zagadnienie synchronizacji procesów, ze względu na to, że stanowi przykład szerokiej klasy problemów sterowania współbieżnością.
- Jest prostym odzwierciedleniem konieczności przydzielania wielu zasobów do wielu procesów w sposób grożący zakleszczeniem lub głodzeniem

Zakleszczenie

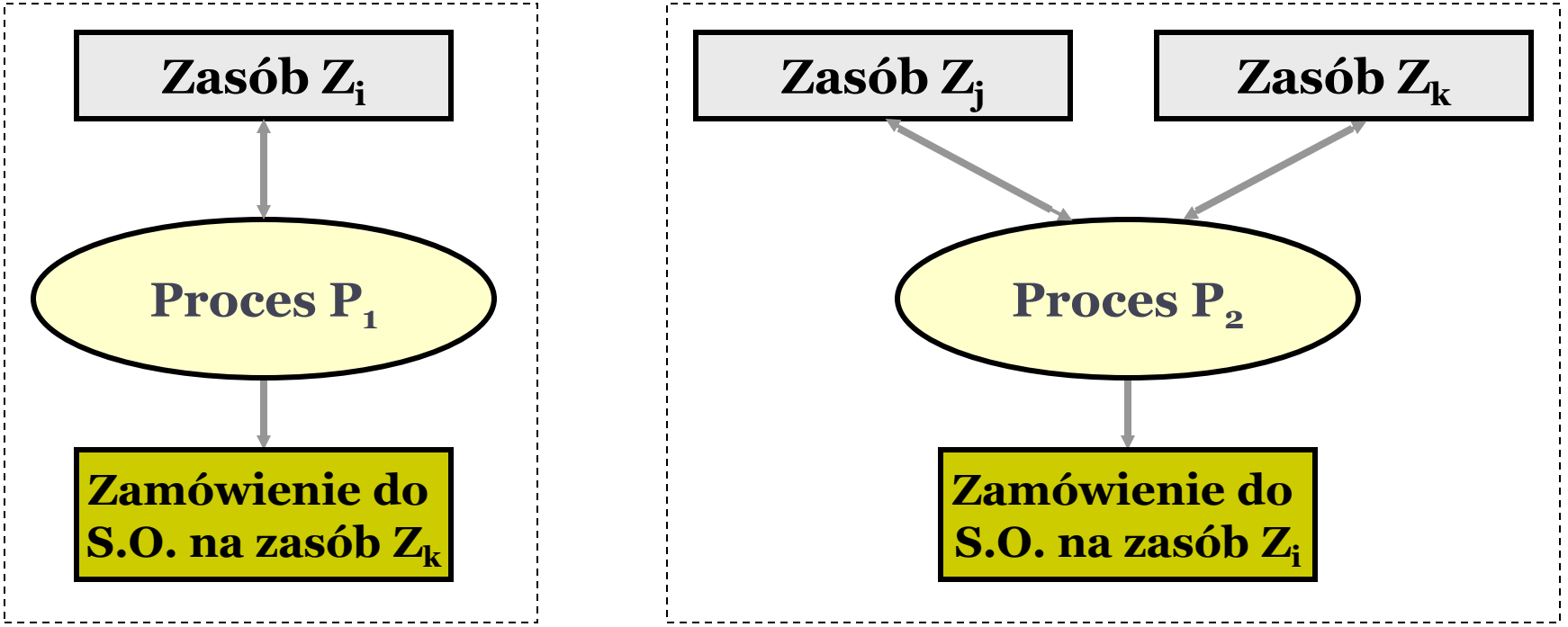
- Istnie kilka metod pozwalających uniknąć zakleszczenia:
  - Pozwolić zasiadać do stołu najwyżej 4 filozofom;
  - Pozwolić na podnoszenie filozofom pałeczek tylko wtedy, gdy obie są wolne;
  - Rozwiązanie asymetryczne:
    - Filozof nieparzysty podnosi najpierw pałeczkę po lewej stronie a później po prawej;
    - Filozof parzysty rozpoczyna od pałeczki z prawej strony i zwraca się do lewej;

Blokada procesów

Problem blokady procesów

- W środowisku wieloprogramowym kilka procesów może rywalizować o *skończoną* liczbę zasobów. Proces zamawia zasób i uzyskuje dostęp, gdy zasób jest wolny, albo czeka na jego zwolnienie przez inny proces.
- Zdarzyć się może taka sytuacja, że procesy nie uzyskają dostępu, bo oczekują na zasoby wzajemnie przetrzymywane, co powoduje, że procesy mogą nigdy nie uzyskać dostępu do zamawianych zasobów.
- Sytuacja, w której procesy nie mogą uzyskać dostępu do zamówionych zasobów, a tym samym zakończyć pracę, nazywa się blokadą, zakleszczeniem (**Deadlock**).

Problem blokady procesów



Czy zasób Z_k jest wolny?
Tak: Przydziel zasób procesowi P_1.
Nie: Proces P_1 czeka na zasób Z_k.

S.O.

Czy zasób Z_i jest wolny?
Tak: Przydziel zasób procesowi P_2.
Nie: Proces P_2 czeka na zasób Z_i.

Warunki konieczne blokady

1. **Wzajemne wyłączenie.**
   Przynajmniej jeden zasób musi być niepodzielny. Zamówienie tego zasobu wymusi konieczność oczekiwania procesu na jego zwolnienie.
2. **Przetrzymywanie i oczekiwanie.**
   Musi istnieć proces przetrzymujący jeden zasób i oczekujący na przydział jakiegoś zasobu, który jest przetrzymywany przez inny proces.
3. **Brak wywłaszczeń.**
   Zasoby nie podlegają wywłaszczeniu, co oznacza, że zwolnienie zasobu może nastąpić jedynie z inicjatywy procesu przetrzymującego zasób.
4. **Czekanie cykliczne.**
   Istnieje zbiór procesów {P0, P1, …, Pn}, takich że P0 czeka na zasób przetrzymywany przez P1, P1 czeka na zasób przetrzymywany przez P2, itd., zaś Pn czeka na zasób przetrzymywany przez P0.

Grafy przydziału zasobów

Graf przydziału zasobów jest grafem zorientowanym G = < *W*, *K* >, gdzie *W* – zbiór wierzchołków grafu, zaś *K* – zbiór krawędzi grafu. Zbiór *W* wierzchołków grafu składa się z dwóch podzbiorów – podzbioru procesów *P* = $\{P_0, P_1, \ldots, P_n\}$ oraz podzbioru zasobów *Z* = $\{Z_1, Z_2, \ldots, Z_m\}$.

1. Krawędź grafu skierowana od procesu P_i do zasobu Z_j (***krawędź zamówienia***) oznacza, że proces P_i zamówił zasób Z_j i czeka na ten zasób.
2. Krawędź grafu skierowana od zasobu Z_j do procesu P_i (***krawędź przydziału***) oznacza, że zasób Z_j został przydzielony procesowi P_i.
3. Kiedy proces P_i zamawia egzemplarz zasobu Z_j, wówczas ***krawędź zamówienia*** umieszcza się w grafie. Gdy zasób zostanie przydzielony, ***krawędź zamówienia*** zostaje zamieniona na ***krawędź przydziału***.
4. Gdy proces zwolni zasób, to z grafu zostaje usunięta ***krawędź przydziału***.

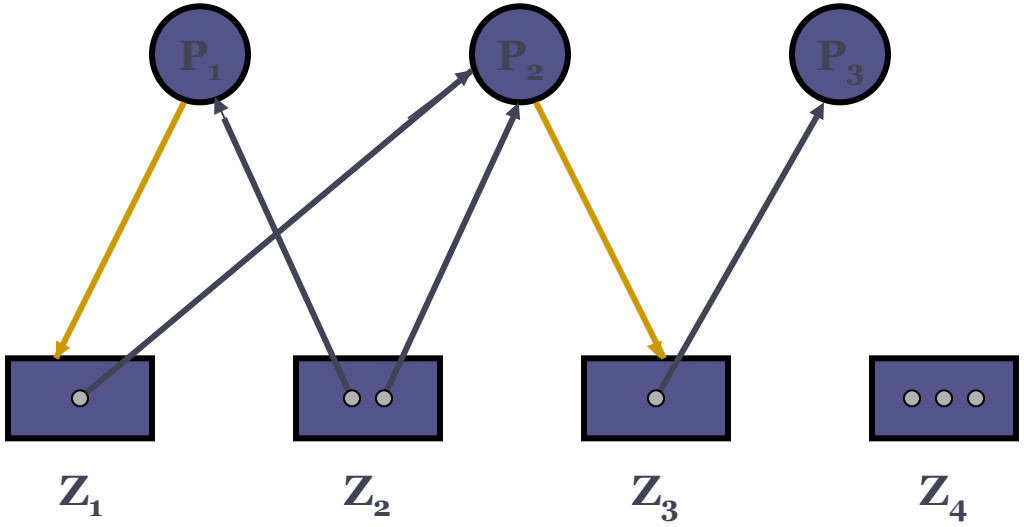
Grafy przydziału zasobów

Graf przydziału zasobów posiada następujące zbiory:

zbiór $P = \{P_1, P_2, P_3\}$,

zbiór $Z = \{Z_1, Z_2, Z_3, Z_4\}$,

zbiór $K = \{P_1 \rightarrow Z_1, P_2 \rightarrow Z_3, Z_1 \rightarrow P_2, Z_2, \rightarrow P_1, Z_2, \rightarrow P_2, Z_3 \rightarrow P_3\}$.



Bez blokady.

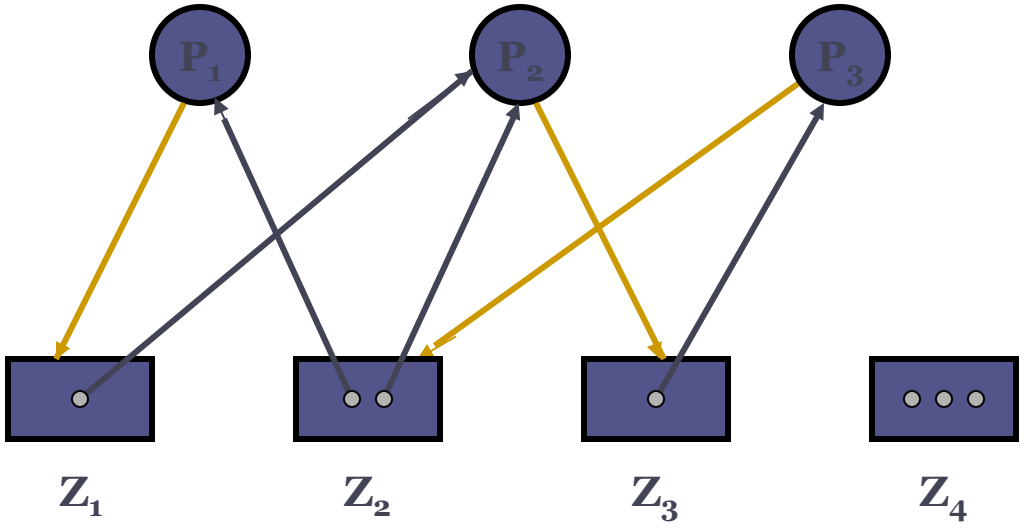
Grafy przydziału zasobów

Graf przydziału zasobów posiada następujące zbiory:

zbiór $P = \{P_1, P_2, P_3\}$,

zbiór $Z = \{Z_1, Z_2, Z_3, Z_4\}$,

zbiór $K = \{P_1 \rightarrow Z_1, P_2 \rightarrow Z_3, P_3 \rightarrow Z_2, Z_1 \rightarrow P_2, Z_2, \rightarrow P_1, Z_2, \rightarrow P_2, Z_3 \rightarrow P_3\}$.



Z blokadą.

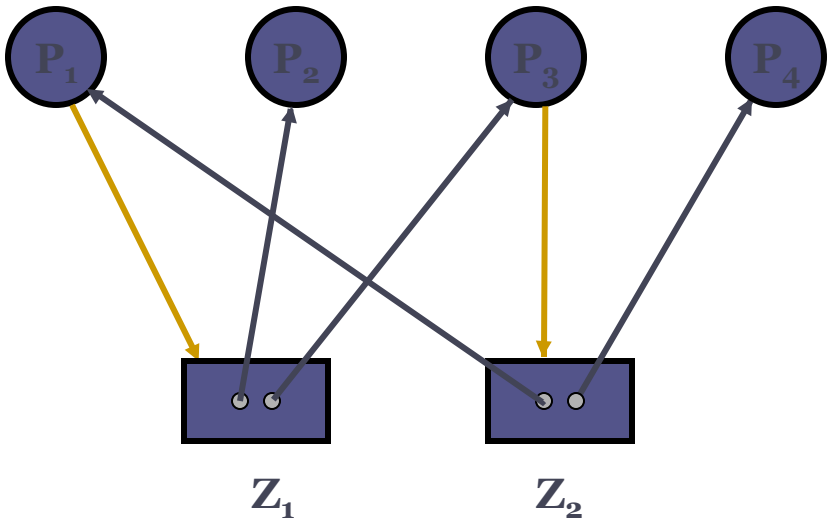
Grafy przydziału zasobów

Graf przydziału zasobów posiada następujące zbiory:

zbiór $P = \{P_1, P_2, P_3, P_4\}$,

zbiór $Z = \{Z_1, Z_2\}$,

zbiór $K = \{P_1 \rightarrow Z_1 , P_3 \rightarrow Z_2, Z_1 \rightarrow P_2, Z_1 \rightarrow P_3, Z_2, \rightarrow P_1, Z_2, \rightarrow P_4,\}$.



Z cyklem, ale bez blokady.

Rozwiązanie problemu blokad

Problem blokady można rozwiązać stosując jedną z następujących strategii:

1. Zapobiegać blokadom w taki sposób, żeby przynajmniej jeden z warunków koniecznych wystąpienia blokady nie był spełniony.
2. Wykrywać powstałe blokady w systemie, a następnie je usuwać.
3. Unikać blokad poprzez wykonywanie odpowiednich czynności uprzedzających możliwość wystąpienia blokady.

Wykrywanie i usuwanie powstałych blokad jest znacznie trudniejsze oraz kosztowniejsze niż zapobieganie blokadom, bądź unikanie blokad. Dlatego generalną zasadą pracy S.O. powinno być zapobieganie blokadom lub ich unikanie. Wykrywanie i usuwanie blokad może wymagać interwencji operatora systemu.

Zapobieganie blokadom

Aby blokada nie wystąpiła wystarcza, żeby nie był spełniony przynajmniej jeden z warunków koniecznych blokady.

1. **Wzajemne wyłączanie** – warunek ten dotyczy przede wszystkim zasobu, który jest niepodzielny, stąd niemożliwe jest zapobieganie blokadom poprzez wykluczenie tego warunku.
2. **Przetrzymywanie i oczekiwanie** – warunek ten dotyczy procesów, stąd można warunek ten uwzględniać, bądź na etapie wprowadzania procesu, bądź na etapie realizacji procesu.
   - Na etapie wprowadzania procesu zamawia się wszystkie zasoby, z których proces będzie korzystał.
   - Proces może przedstawić zamówienie na jakiś zasób, wtedy gdy zwolni wszystkie dotychczas wykorzystywane.

Zapobieganie blokadom

3. **Brak wywłaszczeń** – warunek dotyczy zasobów przydzielonych do procesów. Są dwie metody postępowania:
   - Gdy proces posiadający zasoby zgłasza zapotrzebowanie na inny zasób, wówczas traci (w sposób niejawny) wszystkie zasoby, które były jemu przydzielone. Proces zostanie wznowiony, gdy odzyska wszystkie swoje zasoby oraz zasób zamówiony.
   - Gdy proces zamawia zasoby, wówczas uzyskuje dostęp, gdy zasób jest wolny. Gdy zamawiany zasób jest zajęty, wówczas z puli swoich zasobów oddaje zasoby, na które jest zamówienie od innych procesów. Proces zostanie wznowiony, gdy odzyska swoje zasoby i uzyska dostęp do zasobu zamawianego.
4. **Czekanie cykliczne** – warunek ten dotyczy procesów, które oczekują na zasoby. Uniknięciem sytuacji cyklicznego czekania jest odpowiedni sposób przydziału zasobów. Porządkujemy zasoby przydzielając im numery i przydzielamy zasób Z_j tylko wtedy gdy $F(Z_j) > F(Z_i)$.

Wykrywanie i usuwanie blokad

W metodzie tej dopuszcza się występowanie blokad. Algorytm wykrywania blokady sprawdza, czy spełniony jest warunek cyklicznego czekania, analizując aktualny graf przydziału zasobów. Jest to procedura dosyć uciążliwa.

Po stwierdzeniu blokady stosuje się jedną z następujących technik:

1. Usuwa się wszystkie procesy uczestniczące w blokadzie.
2. Jeśli dla procesów istnieją punkty kontrolne (Check Points), wówczas procesy cofają się do punktów kontrolnych i rozpoczynają realizację procesów od tych punktów. Może to niekiedy prowadzić do ponownej blokady.
3. Z puli procesów oczekujących usuwa się kolejno procesy tak długo, aż zostanie usunięta blokada. Ważne jest określenie zasad wyboru procesów, żeby koszty usuwania procesów były jak najmniejsze.
4. Zawłaszcza się zasoby procesów uczestniczących w blokadzie, sprawdzając każdorazowo, czy procesy są w blokadzie. Proces, któremu zawłaszczono zasoby musi złożyć zamówienie na zasoby.

Unikanie blokad

Algorytm unikania blokady musi przewidywać prawdopodobieństwo wystąpienia blokady, nie dopuszczając do spełnienia zamówienia, które może spowodować blokadę.

Pozornie, następujący algorytm jest słuszny:

> po otrzymaniu zamówienia na zasób, S.O. tworzy nowy graf stanu i jeżeli jest on bez blokady to akceptuje otrzymane zamówienie.

Algorytm ten nie jest jednak poprawny, gdyż nie słuszna jest przesłanka, że jeśli w momencie przydziału zasobu nie pojawiła się blokada, to nie pojawi się ona w dalszym okresie realizacji procesu (*Dijkstra 1968 r.*).

Aby algorytm unikania blokady był skuteczny musi otrzymać on pewne wcześniejsze informacje o ewentualnych wzorcach przyszłych zdarzeń.

Algorytm bankiera.

Zarządzanie pamięcią operacyjną

Cele zarządzania pamięcią

Współużytkowanie procesora przez kilka procesów wymaga przydziału pamięci operacyjnej poszczególnym procesom. Tym samym procesy współdzielą pamięć operacyjną pomiędzy sobą, przy czym aktywny proces obliczeniowy musi być umieszczony w pamięci. Ze względu na ograniczone zasoby pamięci operacyjnej zarządzanie pamięcią musi pozwalać na efektywne wykorzystanie pamięci (kryteria: koszt oraz szybkość).

Zanim proces stanie się aktywny jest on wprowadzany do pamięci operacyjnej z kolejki wejściowej, która na ogół umieszczona jest na dysku.

Zarządzanie pamięcią dotyczyć powinno takich czynników jak:

- relokacja (przemieszczanie) procesów w pamięci;
- ochrona zawartości pamięci;
- dostęp do obszarów współdzielonych;
- organizacja logiczna i fizyczna pamięci.

Cele zarządzania pamięcią

1. Przemieszczanie procesów w pamięci.
   W systemie wieloprogramowym pamięć jest rozdzielona pomiędzy wiele procesów, których liczba może się zmieniać, stąd nie są z góry znane adresy przydzielone poszczególnym procesom. S. O. jest odpowiedzialny za przekształcenie adresów użytych w programie na adresy rzeczywiste.
2. Ochrona zawartości pamięci.
   Gdy kilka procesów dzieli pamięć, wówczas niezwykle istotne staje się zapewnienie nienaruszalności poszczególnych obszarów pamięci. Dotyczy to zarówno pisania do pamięci, jak i czytania (*zachowanie tajemnicy informacji*). S. O. analizuje wszystkie odwołania poszczególnych procesów do pamięci, sprawdzając, czy odwołania dotyczą przydzielonego obszaru pamięci dla procesu.

Cele zarządzania pamięcią

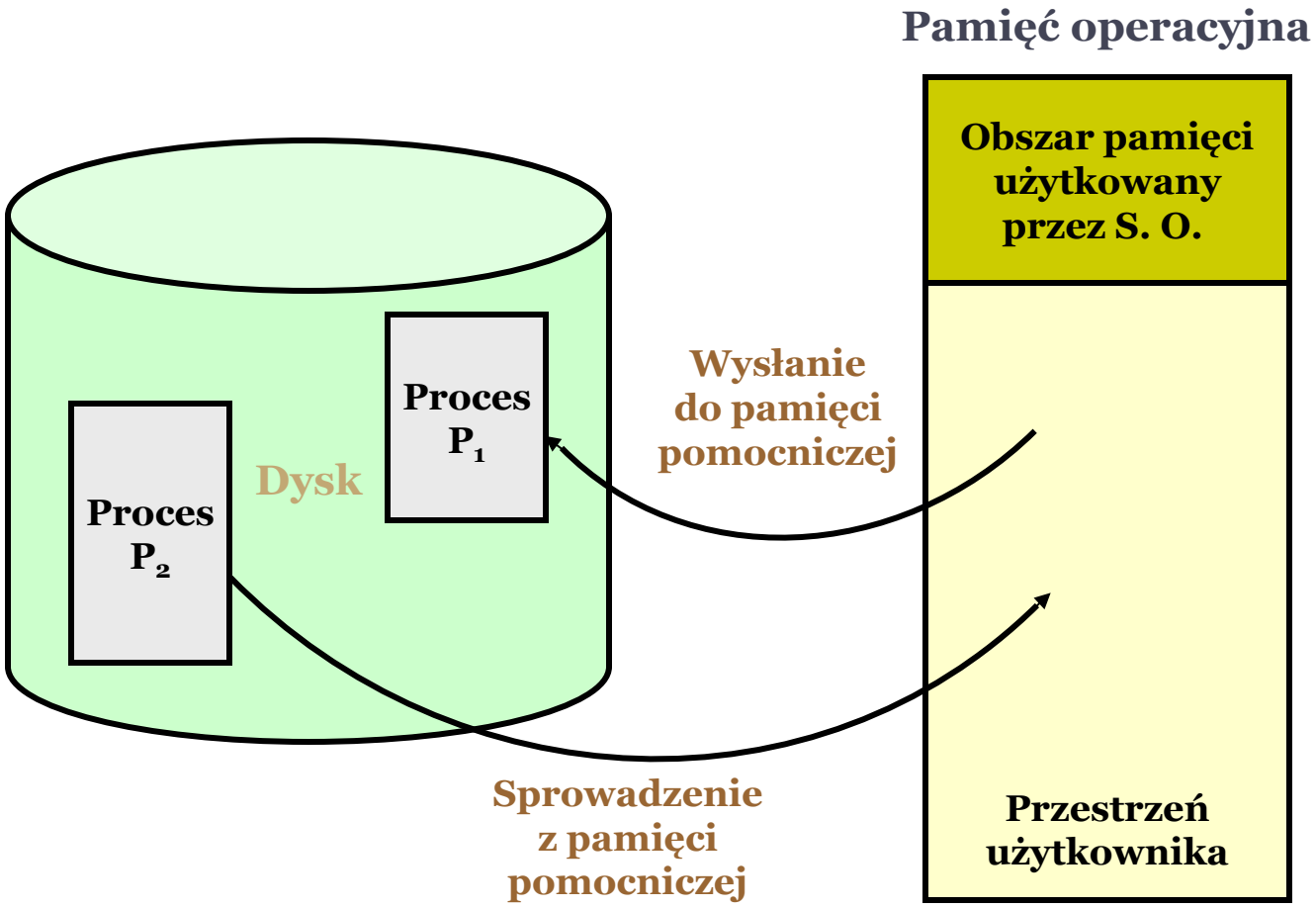
3. Dostęp do obszarów współdzielonych.
   Możliwe są sytuacje, w których umożliwia się kilku procesom dostęp do tych samych obszarów pamięci (np. kilka procesów wykonuje ten sam program, czy procesy współdzielą te same struktury danych). S.O. musi sprawować pełna kontrolę nad dostępem do obszarów dzielonych pamięci realizując przy tym ochronę tych obszarów.
4. Organizacja logiczna pamięci.
   Rzeczywista struktura pamięci jest jednowymiarowa, uporządkowana liniowo. Adresy są ponumerowane od 0 do górnej granicy adresów. Programy mają strukturę złożoną (np. zawierają moduły lub procedury) i dlatego odwołują się do różnych obszarów pamięci, zaś programy i dane znajdują się w różnych obszarach pamięci. Dobrym rozwiązaniem jest segmentacja pamięci. Segmenty można kodować niezależnie, można im wyznaczać różne poziomy ochrony oraz stworzyć mechanizmy współdzielenia.

Cele zarządzania pamięcią

5. Organizacja fizyczna pamięci.

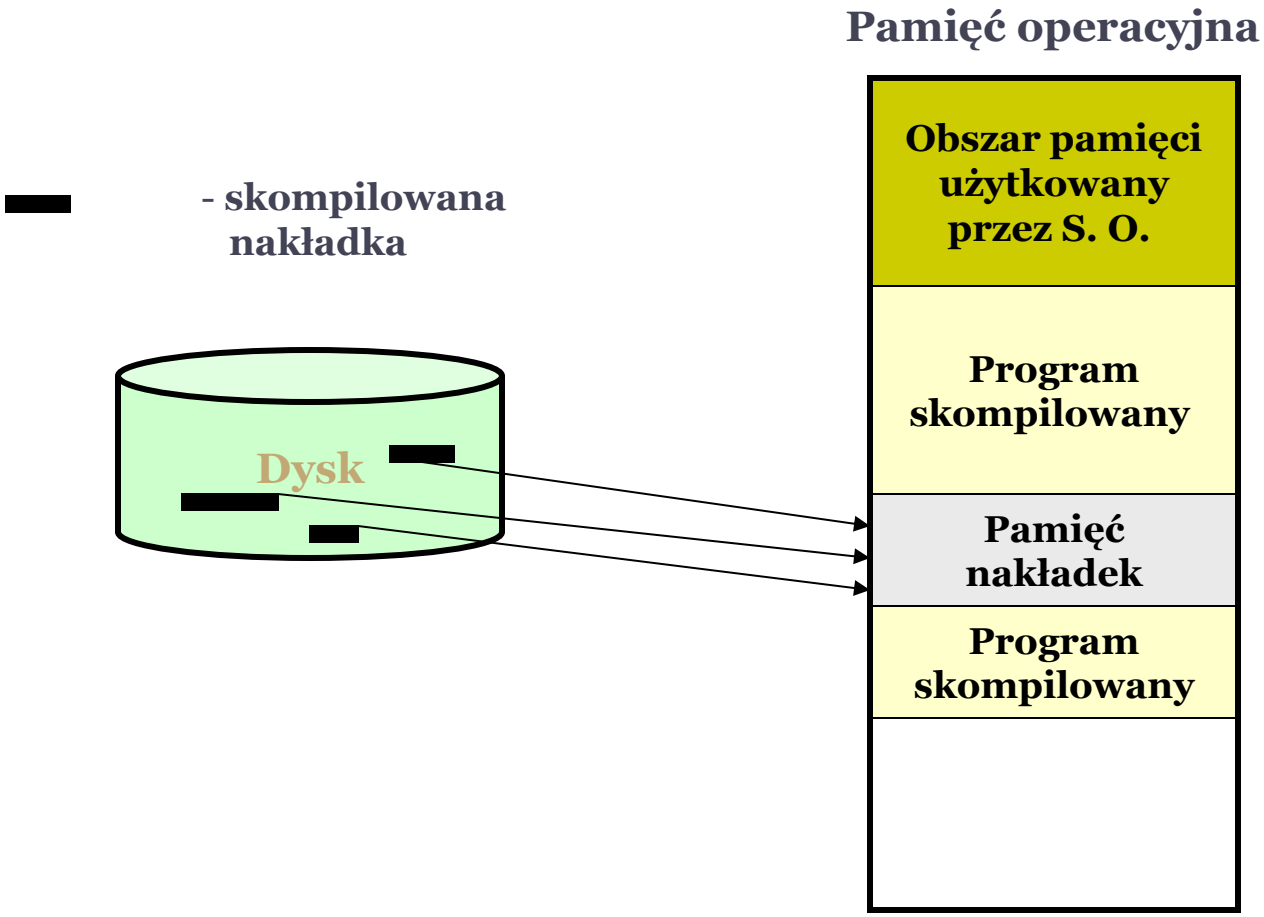
   Ze względu na dosyć niewielkie, fizyczne pojemności pamięci operacyjnej przy jednocześnie dużym zapotrzebowaniu na pamięć, zaistniała konieczność tworzenia dwupoziomowych pamięci: **pamięć operacyjna, pamięć masowa** (np. dysk). W ten sposób uzyskiwano kompromis pomiędzy szybkością pamięci, a jej pojemnością (np. czas dostępu do pamięci półprzewodnikowej był rzędu 70 ns, zaś do dysku 50 ms). Najistotniejszym problemem dwupoziomowych pamięci stała się organizacja wymiany danych pomiędzy obydwoma pamięciami. Początkowo problem wymiany był obsługiwany przez programistów (nakładki– *Overlay*), ale ze względu na wady takiego rozwiązania dosyć szybko zrezygnowano z wymiany danych poprzez techniki nakładkowe. Obecnie, wymianę danych pomiędzy pamięcią masową i operacyjną organizuje system operacyjny tworząc wirtualną organizację pamięci.

Wymiana danych w pamięci



Istotny czynnik – czas przełączania kontekstu.

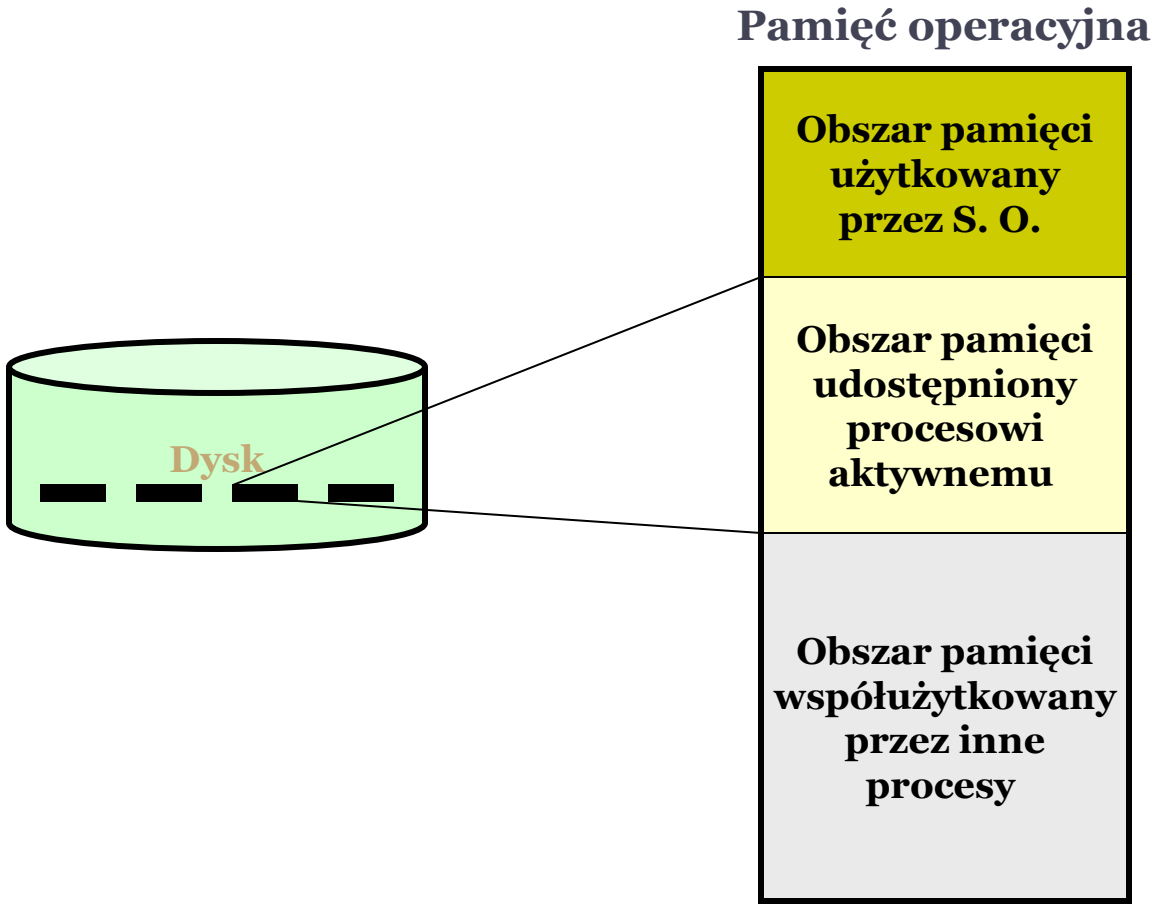
Mechanizm nakładek



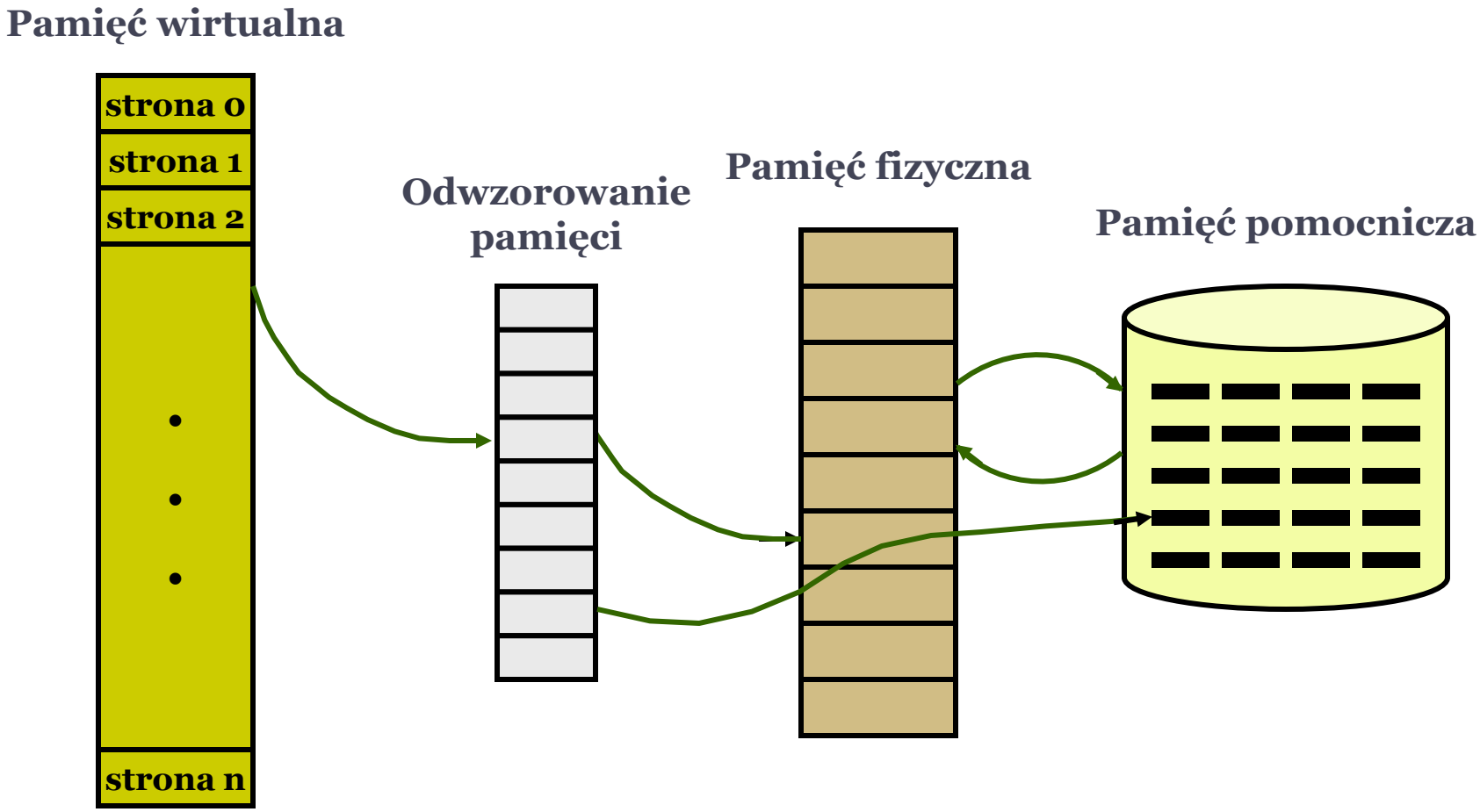
Pamięć wirtualna

1. Wirtualizacja pamięci pozwala programiście widzieć znacznie większą przestrzeń adresową (*przestrzeń nazw*) aniżeli wynosi rzeczywista przestrzeń adresów. Oznacza to, że przestrzeń adresów nie musi być liniowa.
2. Aby taka możliwość istniała niezbędne jest zdefiniowanie mechanizmu, który pozwala na zamianę logicznej przestrzeni adresowej widzianej przez programistę na fizyczną przestrzeń adresów w pamięci komputera.
3. Mechanizm zamiany logicznej przestrzeni adresowej widzianej przez programistę w fizyczną przestrzeń pamięci dostępną na komputerze pamięci pozwala programiście dysponować obszarem przestrzeni adresów zupełnie nie związanym z rzeczywistą wielkością przestrzeni pamięci, czyli pamięcią wirtualną.

Wirtualizacja pamięci



Pamięć wirtualna

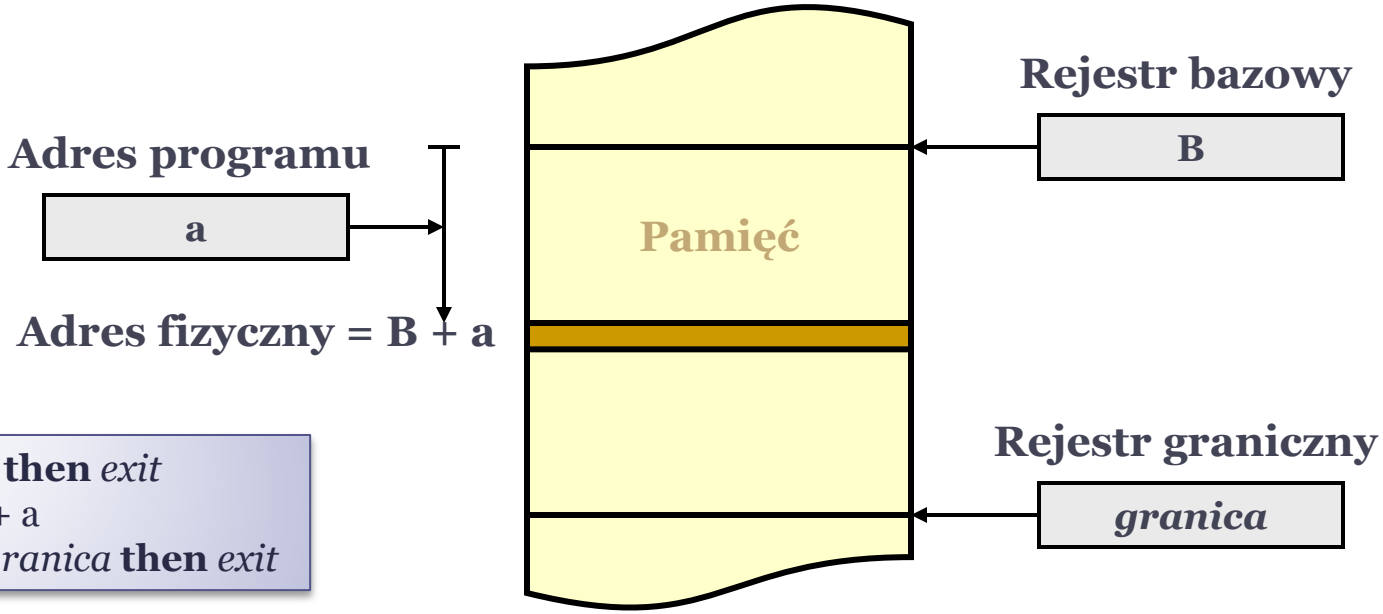


Rejestr bazowy i graniczny

Implementacja pamięci wirtualnej.

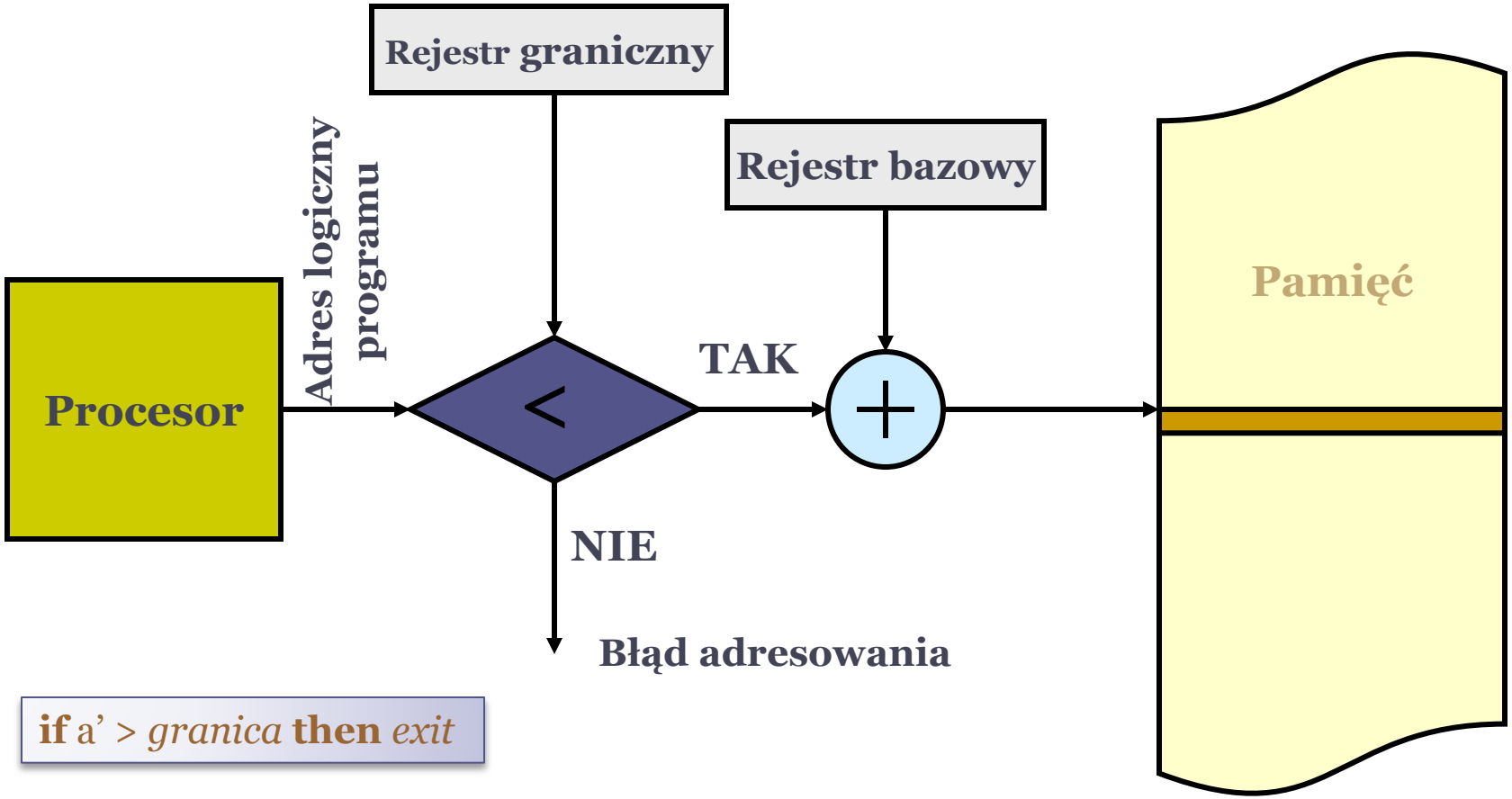
Przemieszczenie i ochronę pamięci realizuje odwzorowanie adresów wykorzystujące:

1. rejestr bazowy zawierający najmniejszy adres procesu;
2. rejestr graniczny określający najwyższy adres pamięci, który może używać proces.

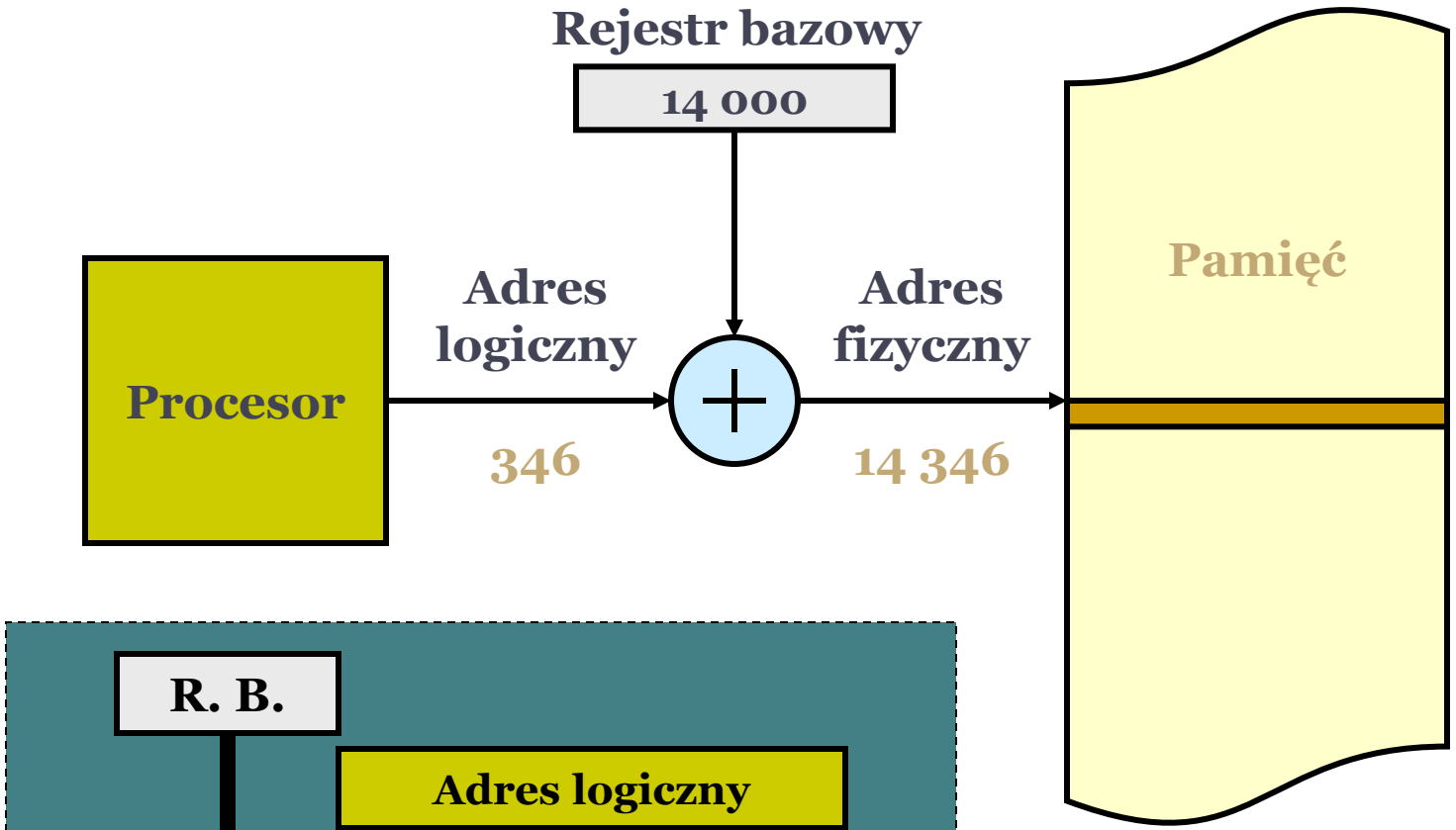


Rejestr bazowy i graniczny

Rejestr graniczny określa dostępny obszar pamięci.



Adresy fizyczne pamięci



Stronicowanie pamięci

Rejestry bazowy i graniczny zapewniają swobodę w wykorzystaniu pamięci, ale przestrzeń adresów jest równa przestrzeni pamięci.

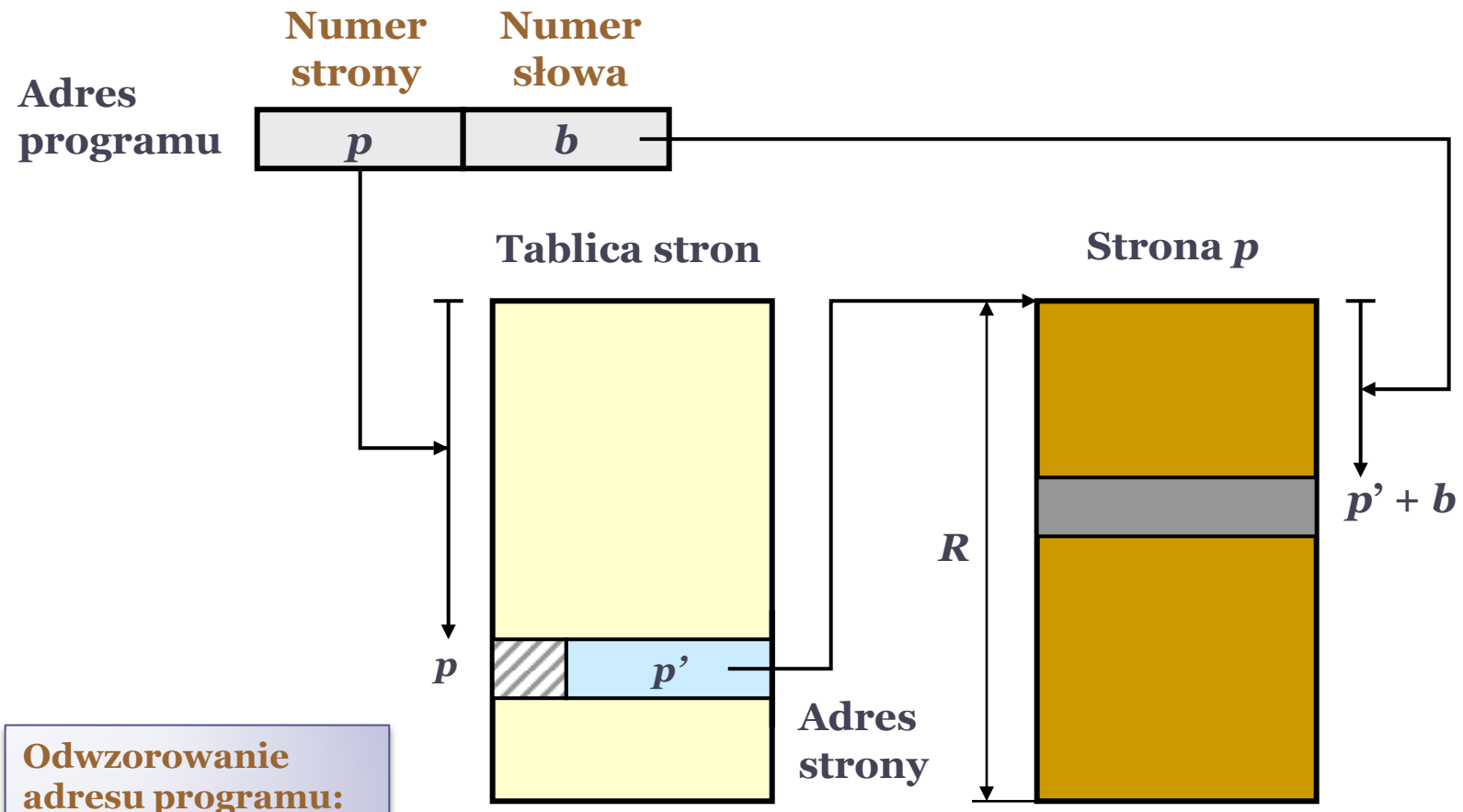
Rozwiązaniem pozwalającym na zwiększenie przestrzeni adresów jest wykorzystanie mechanizmu stronicowania. Przestrzeń adresów wirtualnych jest podzielona na strony (moduły adresów o jednakowej wielkości), zaś pamięć jest podzielona na ramki stron (moduły pamięci o wielkości strony adresów wirtualnych).

Strony procesu mogą być aktywne – ramki stron umieszczone w pamięci, lub nieaktywne – ramki stron umieszczone w pamięci pomocniczej.

Mechanizm stronicowania realizuje dwa zadania:

1. odwzorowuje adresy procesu w numer strony oraz numer słowa na stronie (ramka strony może, ale nie musi być w pamięci);
2. przesyła strony pomiędzy pamięcią główną i pomocniczą.

Stronicowanie pamięci

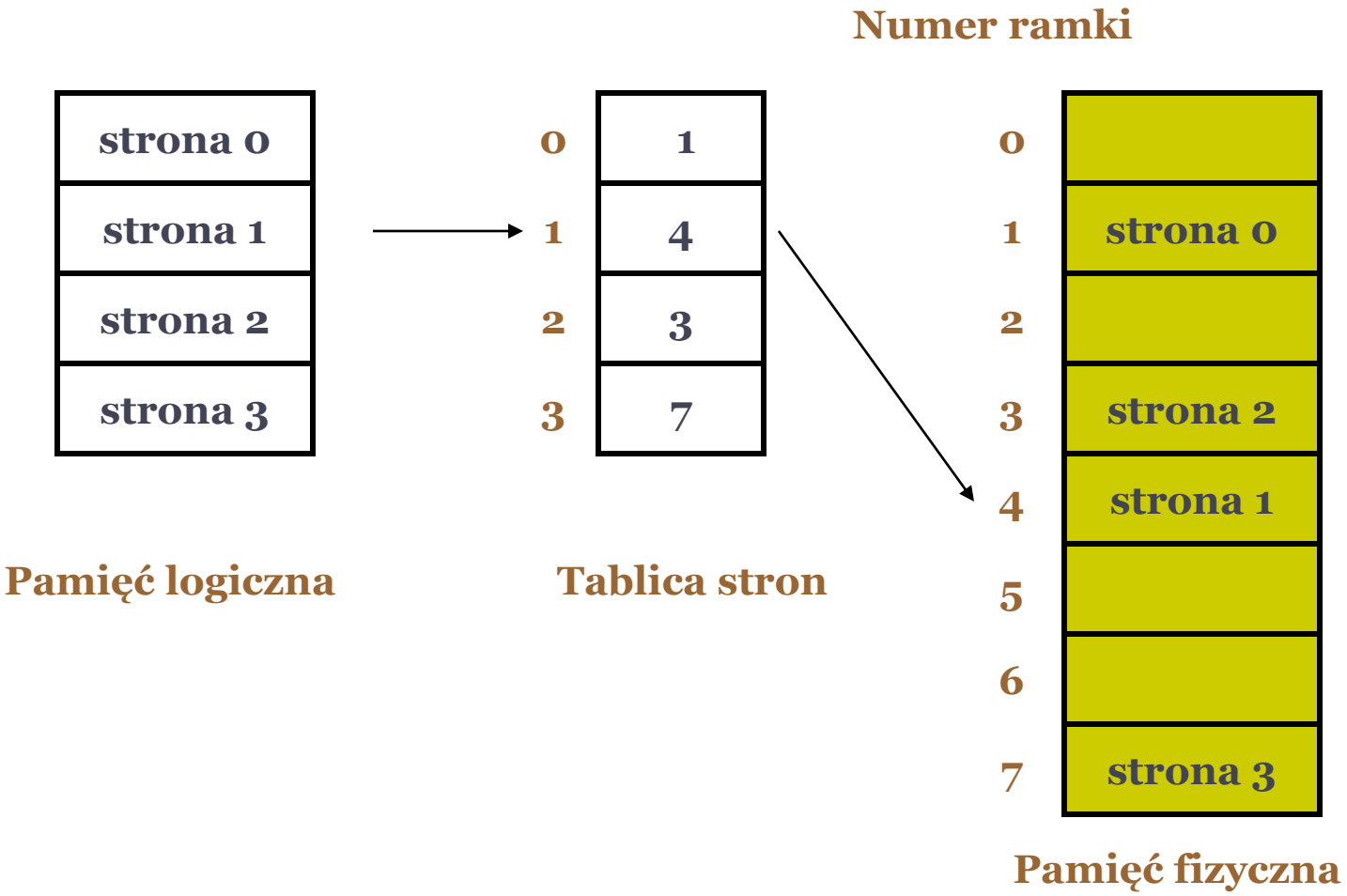


Odwzorowanie adresu programu:

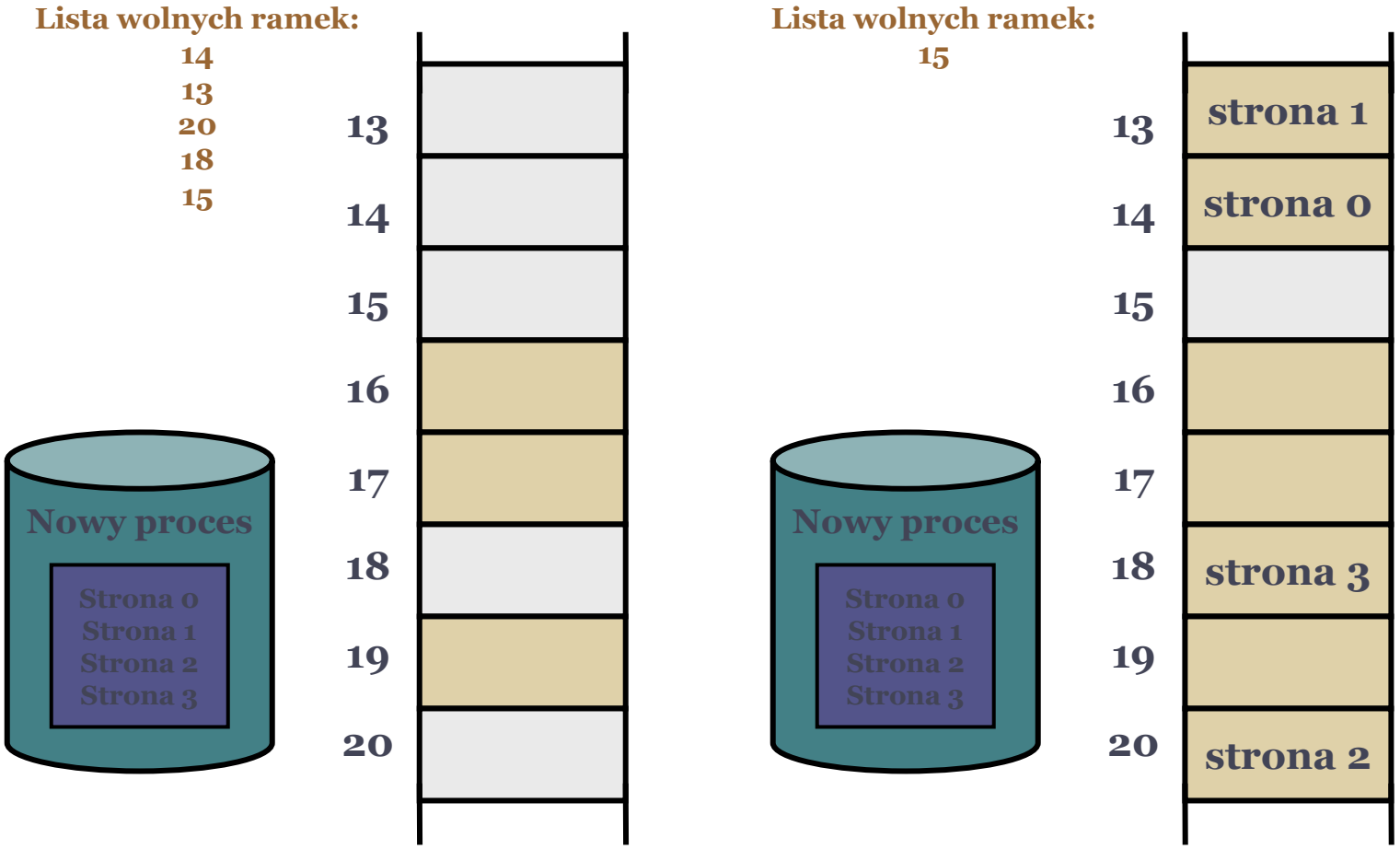
$f(a) = f(p,b) = p' + b$

Numer strony p = E(a/R), numer słowa b = a mod R.

Stronicowanie pamięci



Stronicowanie pamięci

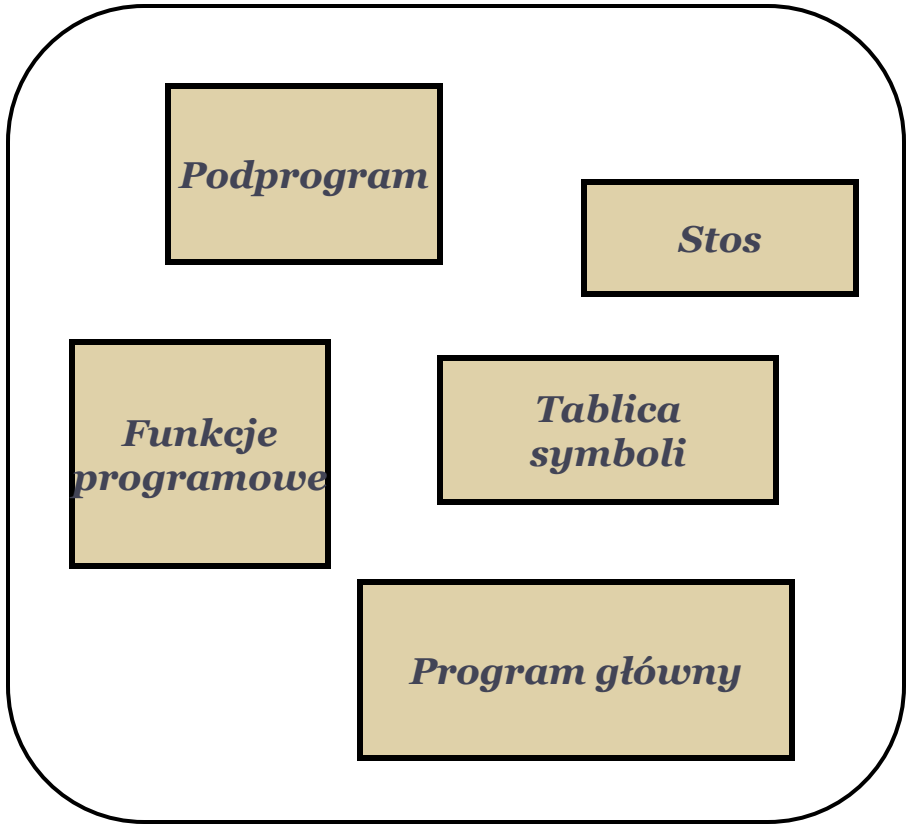


a. Wolne ramki przed przydziałem.

b. Wolne ramki po przydziale.

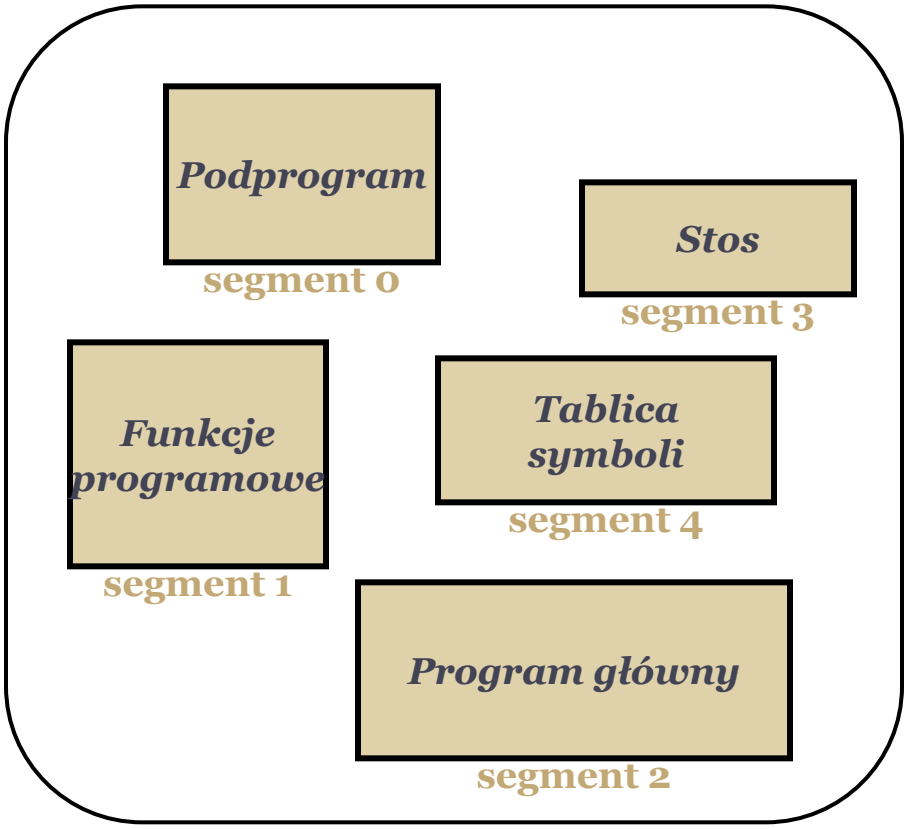
Segmentacja

Program z punktu widzenia użytkownika.



Przestrzeń adresów logicznych

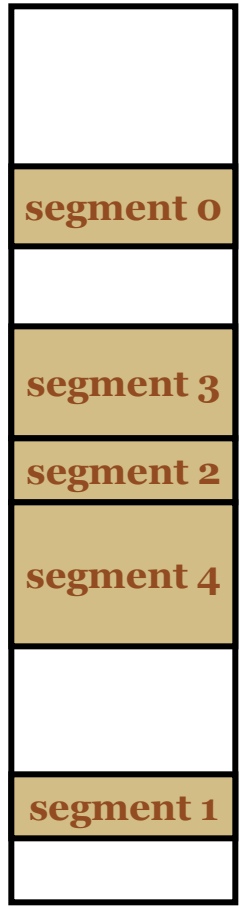
Segmentacja



Przestrzeń adresów logicznych

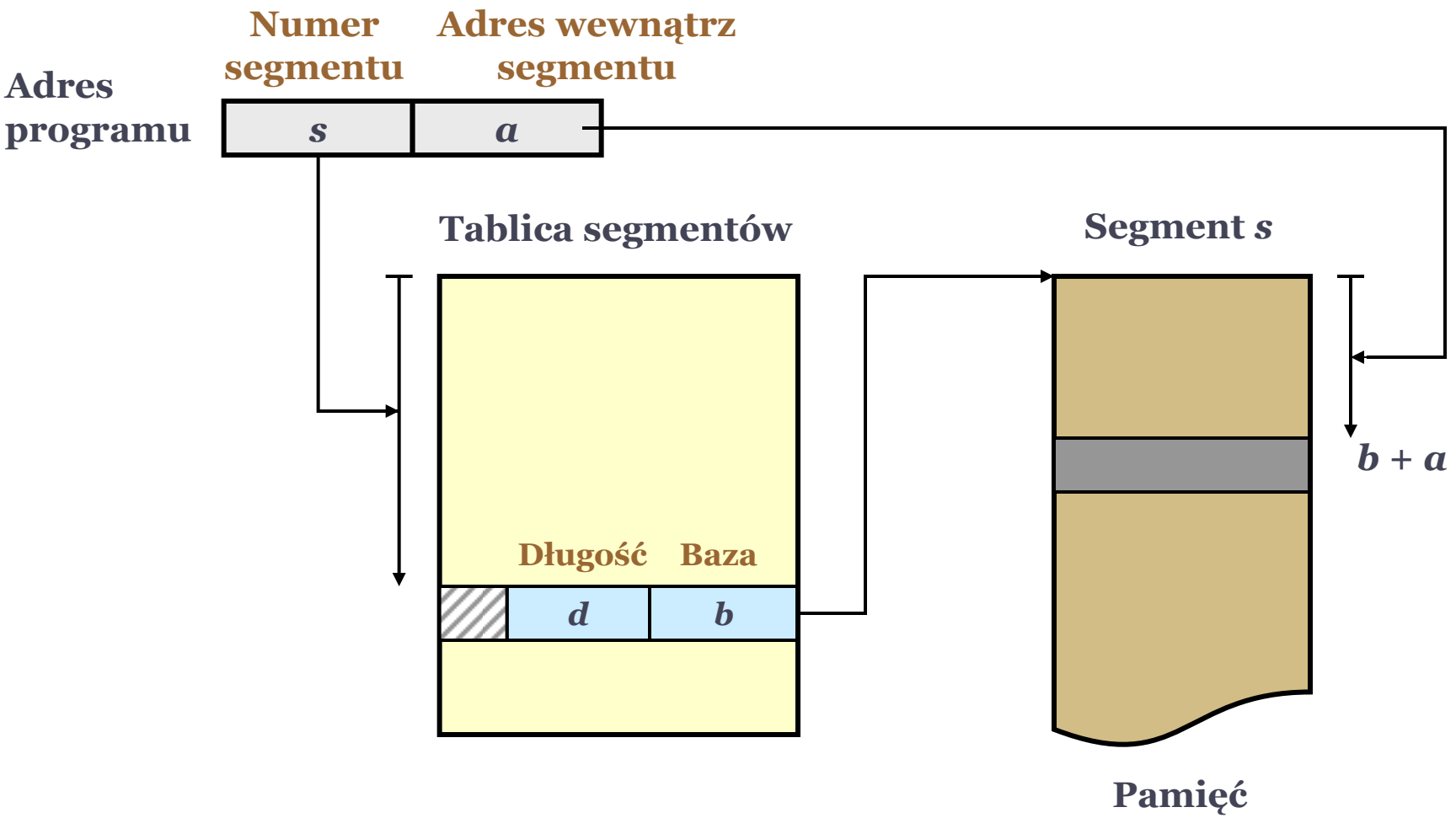
	rozmiar	baza
0	1000	1400
1	400	7500
2	400	4700
3	1500	3200
4	2000	5500

Tablica segmentów



Pamięć fizyczna

Segmentacja



Stronicowanie a segmentacja

1. Celem segmentacji jest logiczny podział przestrzeni adresów, podczas gdy celem stronicowania jest fizyczny podział pamięci, którą chcemy implementować jako pamięć na tym samym poziomie.
2. Strony mają ustalony rozmiar wynikający z architektury komputera, podczas gdy rozmiar segmentów może być dowolny, określony przez programistę.
3. Podział adresu programu na numery strony oraz numer bajtu jest wykonywany środkami sprzętowymi, przy czym przekroczenie zakresu numeracji bajtu powoduje zwiększenie numeru strony.
4. Podział adresu programu na numery segmentu i bajtu jest logiczny, zaś przekroczenie zakresu dla numeru bajtu nie ma żadnego wpływu na numer segmentu (przekroczenie zakresu powoduje jedynie sygnalizację przekroczenia zakresu).

Przydział pamięci

Strategie związane z przydziałem obszarów fizycznej pamięci obejmują trzy następujące kategorie:

1. Strategie wymiany (*Replacement Policies*), które służą do określania tych danych w pamięci, które mogą być usunięte w celu uwolnienia pamięci dla innych danych.
2. Strategie pobierania (*Fetch Policies*), które określają sytuacje, dla których informacje przechowywane w pamięci pomocniczej mają być załadowane do pamięci głównej (np. na żądanie czy z wyprzedzeniem).
3. Strategie rozmieszczania (*Placement Policies*), które służą do wyboru miejsca (miejsc) w pamięci głównej, do którego (których) ma (mają) być załadowana informacja pobierana z pamięci pomocniczej.

Strategie wymiany

Strategie wymiany dla systemów ze stronicowaniem:

1. Najdawniej używana strona. Zakłada się, że zachowanie procesu w przyszłości będzie zupełnie identyczne jak zachowanie dotychczasowe. Strategia ta wymaga stałej rejestracji odwołań.
2. Najrzadziej wykorzystywana strona. Strategia ta wynika z podobnej argumentacji jak w przypadku strategii poprzedniej. Wymaga ona prowadzenia licznika wykorzystania strony. Istnieje niebezpieczeństwo, że wymieniane zostaną strony ostatnio wprowadzone.
3. Najdawniej załadowana strona. Jest to najprostsza strategia, bardzo łatwo implementowana w praktycznej realizacji. Istnieje niebezpieczeństwo usunięcia strony często używanej, gdyż mechanizm usuwania stron nie zależy od jej wykorzystania, a jedynie od momentu jej załadowania.

Strategie wymiany

Strategie wymiany dla systemów bez stronicowania.

Ze względu na to, że segmenty w programie mają z reguły różne wielkości, stąd podstawowa strategia wymiany powinna zapewniać przygotowanie obszaru dla umieszczenia segmentu w pamięci. Decyzja o tym, który segment należy przesłać do pamięci pomocniczej jest uzależniona przede wszystkim od rozmiaru segmentu.

W przypadku alternatywnych możliwości (np. usunąć jeden duży segment, czy kilka małych) można się posiłkować strategiami wymiany dla systemów ze stronicowaniem (szczególnie łatwe do implementacji, gdy segmenty są stronicowane). Istnieje niebezpieczeństwo usunięcia segment, do którego za chwilę może być zrealizowane odwołanie.

Strategie pobierania

Strategie pobierania danych z pamięci pomocniczej określają, kiedy należy przesłać blok z pamięci pomocniczej do pamięci głównej.

Strategie pobierania dzielą się na dwie klasy:

strategie pobierania na żądanie (*On Demand*);
strategie pobierania przewidującego (*Anticipatory*).

Łatwiejszą strategią do implementacji jest strategia pobierania na żądanie, gdyż błąd braku bloku (segmentu, strony) inicjuje proces pobierania.

Strategie pobierania przewidującego wymagają opracowania algorytmu, który będzie przewidywał przyszłe zachowanie procesu.

Przewidywanie opiera się na:

- znajomości właściwości konstrukcji programu,
- wnioskowaniu z dotychczasowego przebiegu procesu.

Strategie rozmieszczania

Strategie rozmieszczania danych w pamięci głównej dla systemów bez stronicowania.

Pamięć segmentowana

Strategie rozmieszczania danych wymagają aktualnej listy adresów oraz rozmiarów wszystkich dziur pamięci. Lista ta musi być aktualizowana z każdym wykorzystaniem dziury (dziur).
W przypadku, gdy rozmiar dziury jest większy od rozmiaru segmentu, wówczas segment zajmuje pewien obszar dziury zaś pozostała część staje się dziurą.

a. 

b. 

Strategie rozmieszczania

1. Strategia najlepszego dopasowania.

   Dziury są uporządkowane rosnąco pod względem ich rozmiaru:

   $$x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_n .$$

   Jeżeli s oznacza rozmiar segmentu ładowanego do pamięci, to należy znaleźć takie i, że:

   $$s \leq x_i .$$

2. Strategia najgorszego dopasowania.

   Dziury są uporządkowane malejąco pod względem ich rozmiaru:

   $$x_1 \geq x_2 \geq x_3 \geq \ldots \geq x_n .$$

   Umieść segment w pierwszej dziurze, a dziurę utworzoną przez pozostały fragment umieść w odpowiednim miejscu listy dziur.

Strategie rozmieszczania

3. Strategia pierwszego dopasowania.

   Dziury są uporządkowane rosnąco pod względem ich adresów bazowych.

   $$A(x_1) \leq A(x_2) \leq A(x_3) \leq \ldots \leq A(x_n) .$$

   Znajdź najmniejsze i takie, że dla segmentu o rozmiarze s:

   $$s \leq x_i .$$

   Algorytm grupuje adresy małych dziur na początku listy.

Strategie rozmieszczania

4. Algorytm bliźniaków (*Buddy Algorithm*).

   Algorytm bliźniaków działa na segmentach, których rozmiar s jest postaci $s = 2^i$, dla $i \leq k$. Tworzy się odrębne listy dziur o rozmiarach 2^1, 2^2, 2^3, ... , 2^k. Dziurę można usunąć z listy o numerze $(i + 1)$, wtedy gdy podzieli się ją na dwie połowy o rozmiarze 2^i, tworząc parę bliźniaków. Podobnie, parę bliźniaków można usunąć z listy o numerze i jeżeli połączy się bliźniacze dziury w jedną dziurę o rozmiarze 2^{i+1}.

```
procedure znajdź_dziurę (i);
begin if i = k + 1 then błąd;
  if lista (i) pusta then begin znajdź_dziurę (i+1);
     podziel dziurę na bliźniaki;
     umieść bliźniaki w lista(i);
  end;
  pobierz pierwszą dziurę z lista(i)
end;
```

Zarządzanie pamięcią pomocniczą

Pamięć pomocnicza

Pamięć pomocnicza stanowi rozszerzenie pamięci głównej, w której przechowuje się zarówno dane jak i programy.

- Pojemność pamięci głównej jest za mała, na to żeby przechowywać wszystkie dane i programy.
- Pamięć **główna** jest medium ulotnym, w którym informacje ulegają stracie po wyłączeniu lub zaniku zasilania.

Rolę pomocniczej pamięci **masowej** pełnią obecnie dyski, choć można do tego celu używać również pamięci masowe o dostępie swobodnym RAM dyski.

Dysk składa się z dwóch podstawowych części:

- napędu dysku;
- sterownika (kontrolera) dysku.

Organizacja dysku: ścieżki (cylindry), sektory.

Pamięć pomocnicza

Pamięć dyskowa charakteryzuje się następującymi właściwościami:

- Informacje z dysku mogą być uaktualniane bez zmiany miejsca, co oznacza, że można przeczytać blok z dysku, wprowadzić do niego zmiany oraz zapisać go z powrotem na tym samym miejscu na dysku.
- Dowolny blok na dysku można zaadresować bezpośrednio. Oznacza to, że na dysku możliwy jest zarówno swobodny jak i sekwencyjny dostęp do informacji (do dowolnego pliku). Przełączanie od jednego pliku do innego wymaga jedynie zmiany miejsca głowicy oraz odczekania na obrót dysku.

Na dysku znajdować się musi katalog dysku, w którym przechowywane są informacje o plikach znajdujących się na dysku.

Zarządzanie pamięcią dyskową

Zarządzanie pamięcią pomocniczą (dyskową) związane jest z następującymi elementami:

- z zarządzaniem *wolnymi obszarami* pamięci, czyli określaniem sposobu ewidencjonowania obszarów, które mogą być wykorzystywane do zapisu danych lub programów;
- z zarządzaniem *przydziałem miejsca* na dysku, czyli określaniem metod udostępniania oraz ewidencjonowania obszarów na dysku;
- z *planowaniem* sposobu obsługi zadań do realizacji, rozumianych jako planowanie dostępu do dysku, czyli przydziałem dostępu do określonej ścieżki na dysku.

Zarządzanie wolnymi obszarami

Dla zarządzania wolnymi obszarami (np. po usunięciu nieaktualnych plików) system wykorzystuje specjalną *listę wolnych obszarów*.
Do zarządzania wolnymi obszarami pamięci dyskowej wykorzystuje się następujące mechanizmy:

- *wektory bitowe;*
- *listy powiązane;*
- *grupowanie;*
- *zliczanie.*

Wektor bitowy:

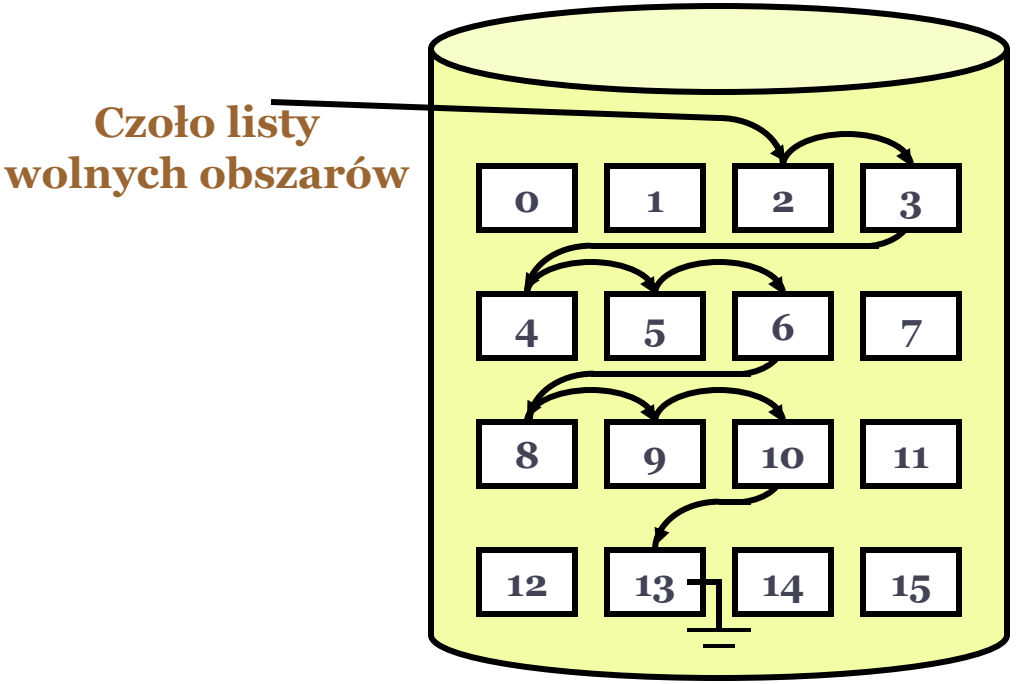
Lista wolnych obszarów jest zapisana w następujący sposób:
blok jest reprezentowany przez 1 bit. Jeśli blok jest wolny, wówczas bit bloku = 0, gdy jest wykorzystywany, wówczas bit bloku = 1.

Przykład: Bloki 2, 3, 4, 7, 8, 12, 14 są wolne, reszta z 16 bloków zajęta.
Wektor bitowy = 1101011100110001.

Zarządzanie wolnymi obszarami

Lista powiązana:

W liście powiązanej przechowywane są wskaźniki wolnych bloków. Metoda ta wymaga przeglądania całej listy w przypadku, gdy zamierzamy znaleźć jakiś blok, lub sprawdzić wszystkie bloki.



Zarządzanie wolnymi obszarami

Grupowanie:

Jest to modyfikacja metody listy powiązanej polegająca na tym, że w pierwszym wolnym bloku listy przechowuje się n adresów. Pierwsze n-1 adresów wskazuje na wolne bloki. Ostatni adres jest adresem dyskowym zawierającym adresy kolejnych n bloków.
Zalety metody: łatwo można znaleźć adresy dużej liczby bloków.

Zliczanie:

Metoda ta wykorzystuje fakt, że kilka kolejnych bloków można przydzielać i zwalniać jednocześnie. Zamiast przechowywać listę n wolnych adresów bloków, przechowuje się adres pierwszego wolnego bloku oraz liczbę n wolnych bloków następujących po sobie. Każda pozycja na liście wolnych obszarów składa się z adresu dyskowego i licznika. Dla liczników > 1 takie rozwiązanie jest efektywniejsze.

Przydział miejsca na dysku

Z reguły na dysku przechowuje się wiele plików. Przydział miejsca na dysku powinien uwzględniać dwa aspekty:

- efektywne wykorzystanie dysku;
- szybki dostęp do plików.

Korzysta się z trzech metod przydziału miejsca na dysku:

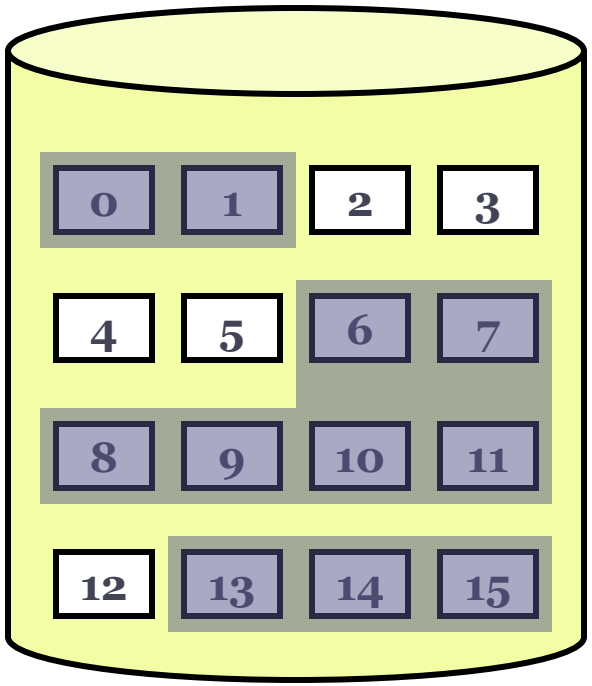
- metody ciągłej;
- metody listowej;
- metody indeksowej.

Ponieważ każda z metod posiada zarówno zalety jak i wady, stąd w niektórych systemach komputerowych stosuje się wszystkie trzy metody jednocześnie.

Przydział miejsca na dysku

Przydział ciągły:

W *metodzie przydziału ciągłego* każdy plik musi zajmować ciąg adresów na dysku, przy czym adresy dyskowe definiują uporząd liniowe.



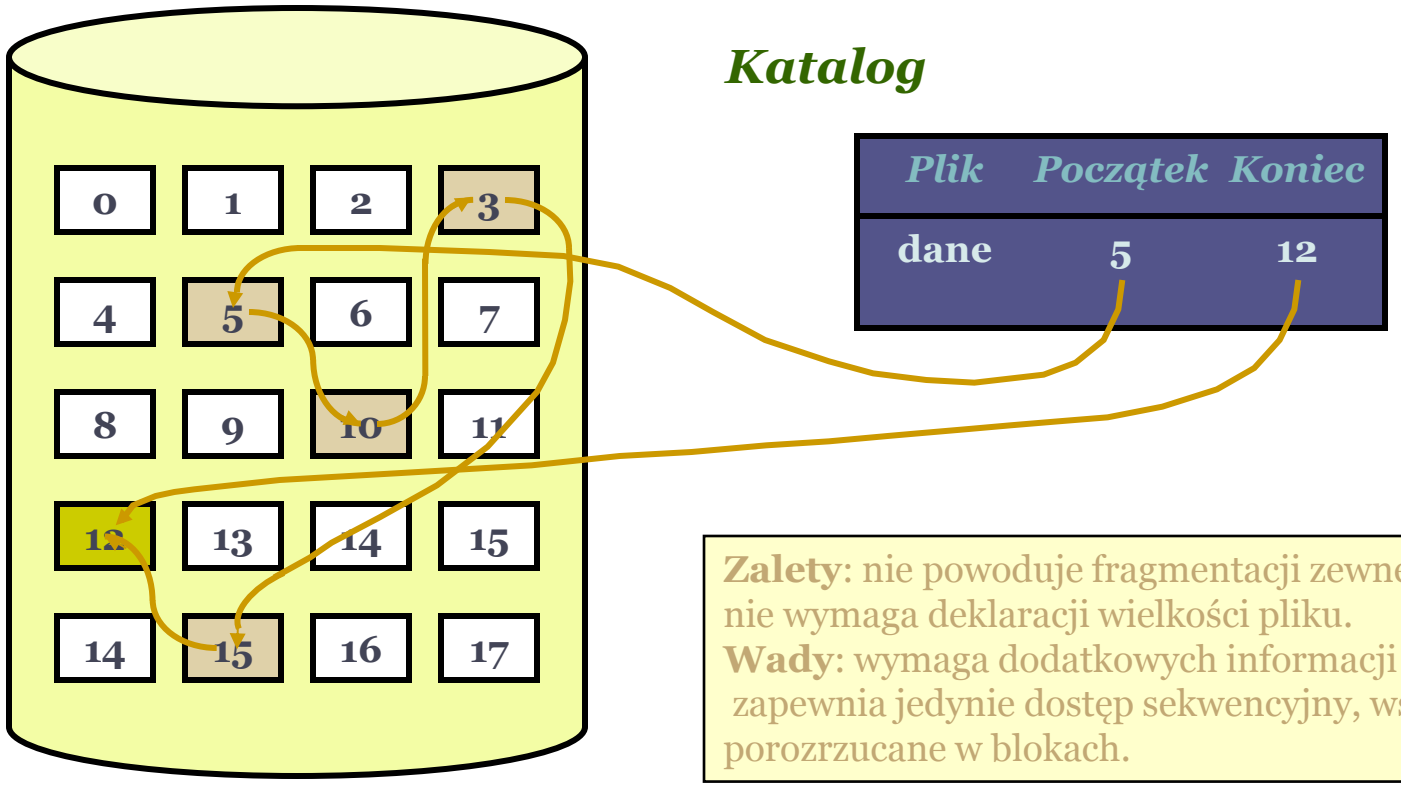
Plik	Początek	Długość
muzyka	6	6
dane	0	2
poczta	13	3

Zalety: może realizować dostęp bezpośredn i sekwencyjny.
Wady: powoduje fragmentację zewnętrzną, wymaga deklaracji rozmiaru pliku.

Przydział miejsca na dysku

Przydział listowy:

W *metodzie przydziału listowego* pliki tworzone są na zasadzie listy.

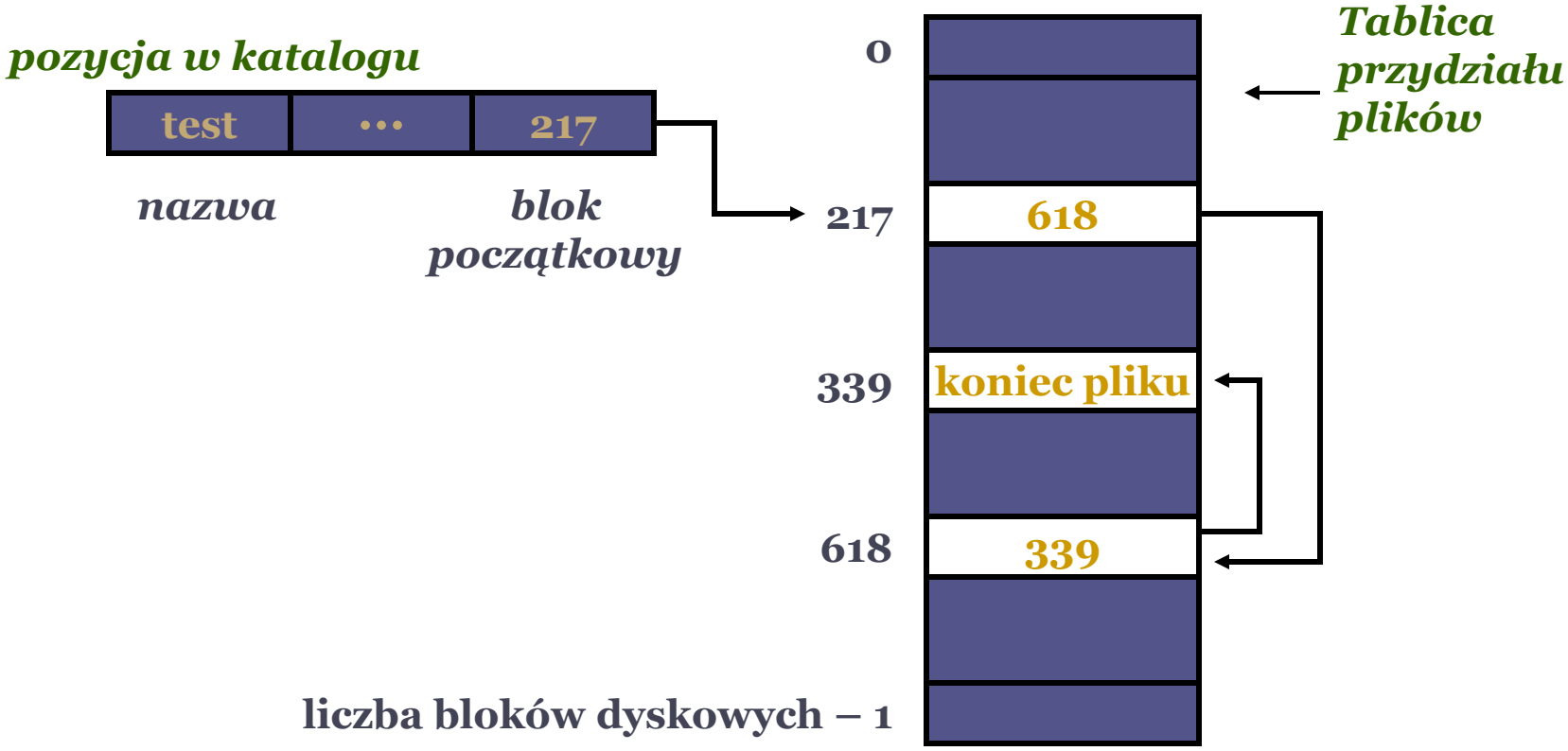


Zalety: nie powoduje fragmentacji zewnętrznej, nie wymaga deklaracji wielkości pliku.
Wady: wymaga dodatkowych informacji (listy), zapewnia jedynie dostęp sekwencyjny, wskaźniki porozrzucane w blokach.

Przydział miejsca na dysku

Przydział listowy (*File-Allocation Table*):

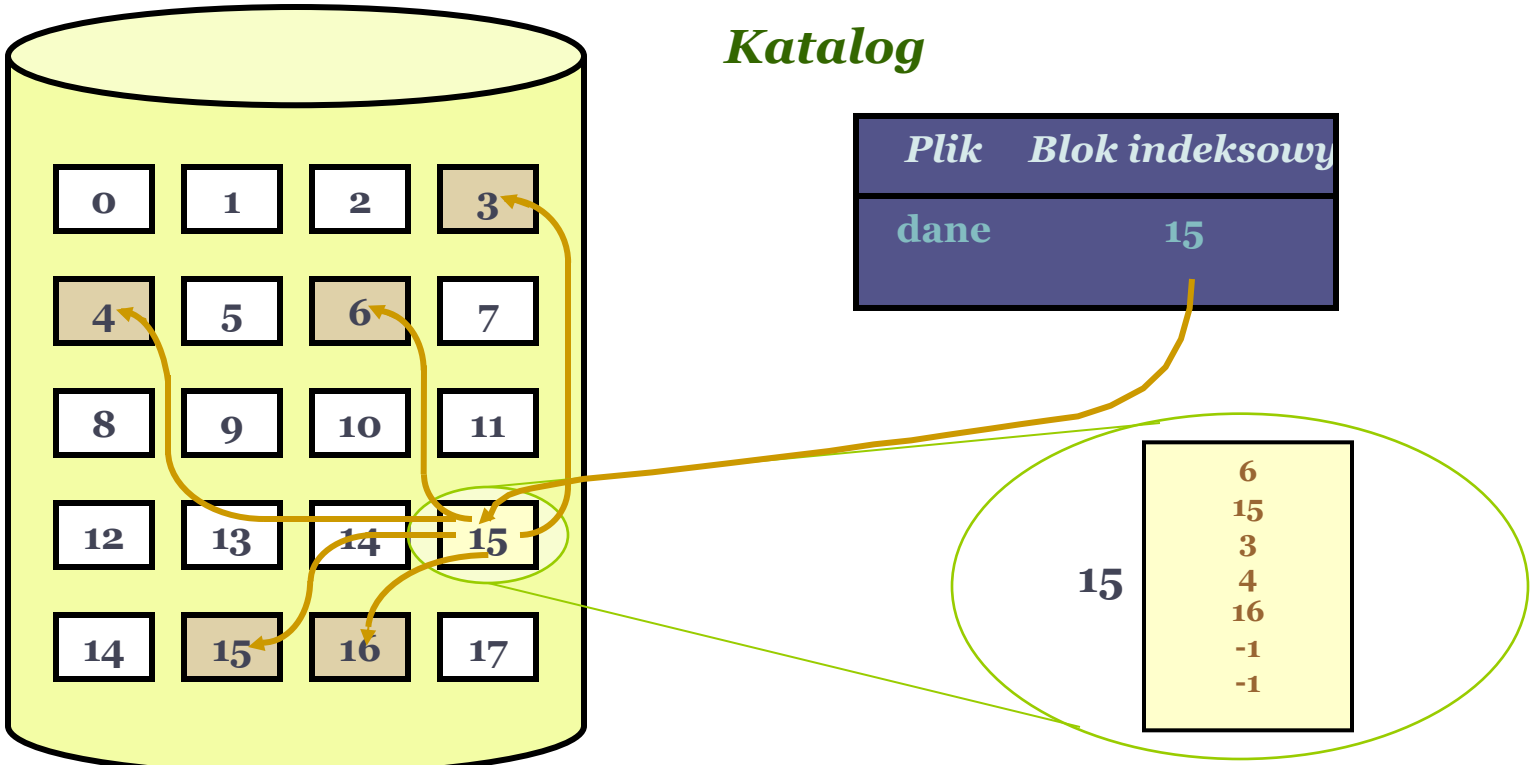
Na dysku wydziela się sekcję zawierającą tablicę, która pełni rolę listy, z jedną pozycją dla bloku oraz indeksowanej numerami bloków.



Przydział miejsca na dysku

Przydział indeksowy:

Dla każdego pliku umieszcza się na dysku blok indeksowy, czyli tablicę adresów bloków dyskowych.



Planowanie dostępu do dysku

Dostęp do dysku decyduje o szybkości pracy systemu. Najistotniejszy jest czas ruchu głowicy, czyli czas przeszukiwania dysku. Rola S. O. polega na zmniejszaniu tego czasu. Dlatego muszą być zdefiniowane odpowiednie metody planowania dostępu do dysku.

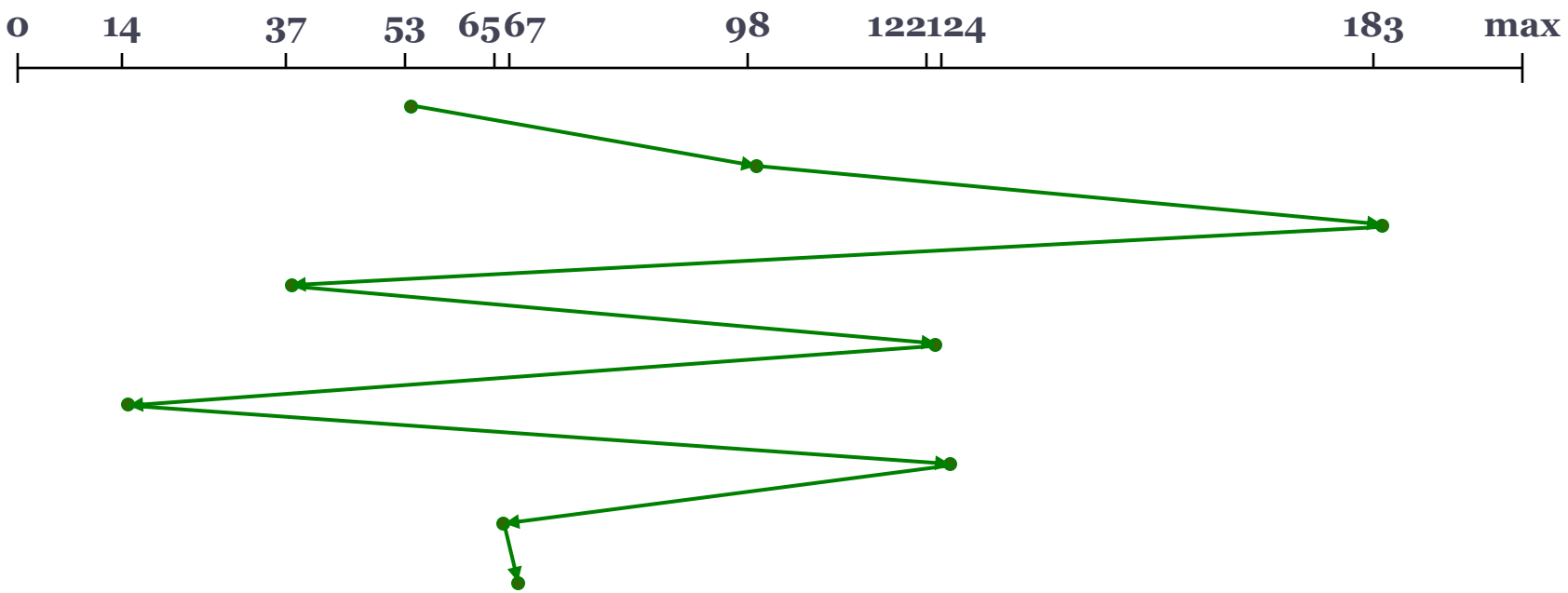
Znane są następujące metody planowania dostępu:

1. Planowanie metodą FCFS (*First Come First Serve*);
2. Planowanie metodą SSFT (*Shortest Seek Time First*);
3. Planowanie metodą SCAN;
4. Planowanie metodą C-SCAN (*Cyclic SCAN*);
5. Planowanie metodą LOOK oraz C-LOOK.

Planowanie metodą FCFS

Niech głowica znajduje się nad ścieżką 53 oraz istnieje uporządkowana kolejka dyskowa ścieżek postaci:

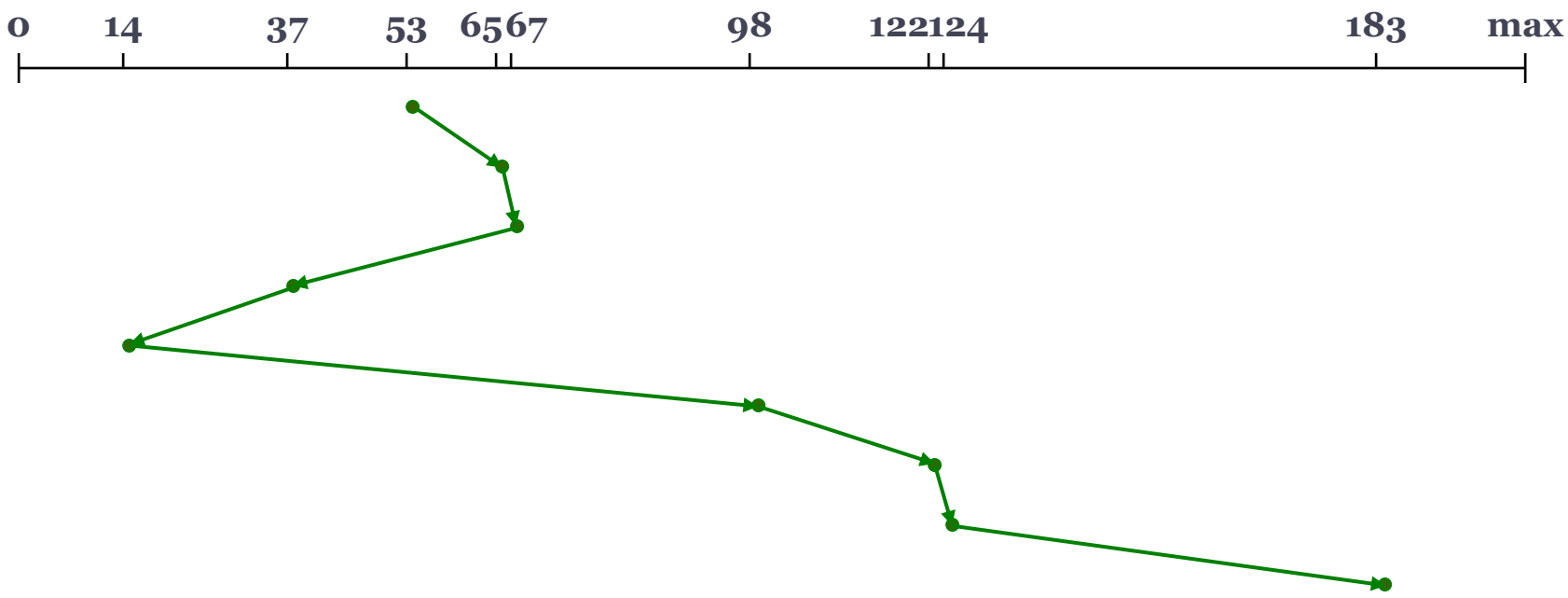
98, 183, 37, 122, 14, 124, 65, 67.



Planowanie metodą SSTF

Niech głowica znajduje się nad ścieżką 53 oraz istnieje uporządkowana kolejka dyskowa ścieżek postaci:

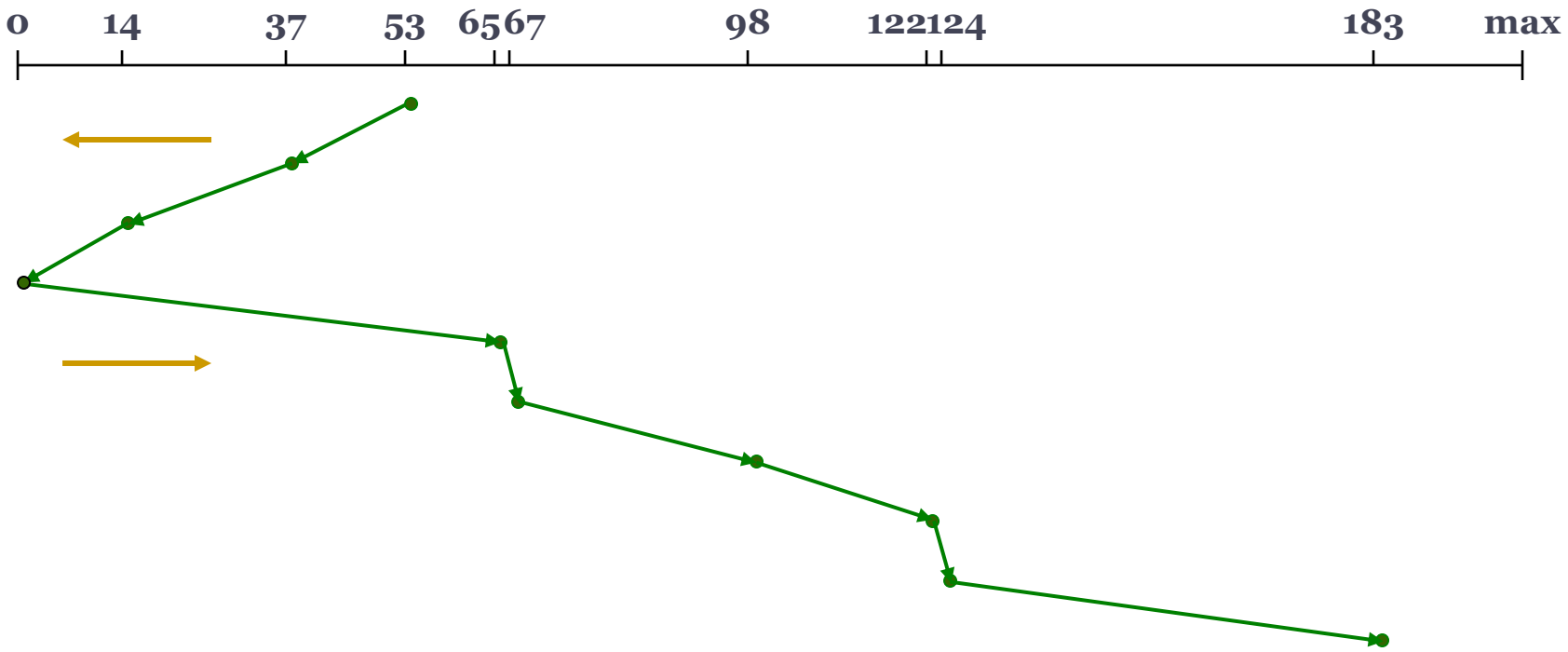
98, 183, 37, 122, 14, 124, 65, 67.



Planowanie metodą SCAN

Niech głowica znajduje się nad ścieżką 53 oraz istnieje uporządkowana kolejka dyskowa ścieżek postaci:

98, 183, 37, 122, 14, 124, 65, 67.



Planowanie metodą C-SCAN

Niech głowica znajduje się nad ścieżką 53 oraz istnieje uporządkowana kolejka dyskowa ścieżek postaci:

98, 183, 37, 122, 14, 124, 65, 67.