Sisteme de Operare

Administrarea memoriei
partea I-a

Cristian Vidrașcu

https://profs.info.uaic.ro/~vidrascu

Cuprins

- Introducere
- Probleme
- Ierarhii de memorie
- Alocarea memoriei
  - Adrese de memorie logice vs. fizice
  - Scheme de alocare contigue
  - Memorie virtuală

  (va urma)
  - Scheme de alocare necontigue

Introducere

- Codul executabil și datele programului trebuie să fie rezidente în memorie pentru a putea fi accesate de către procesor/procesoare
- Este nevoie de *partiționarea* memoriei (pentru a permite utilizarea ei simultan de către mai multe procese) și de *protecția* memoriei (pentru a împiedica procesele să se "deranjeze" unele pe altele)
- Utilizatorii pot ignora modul în care adresele de memorie sunt generate de programe

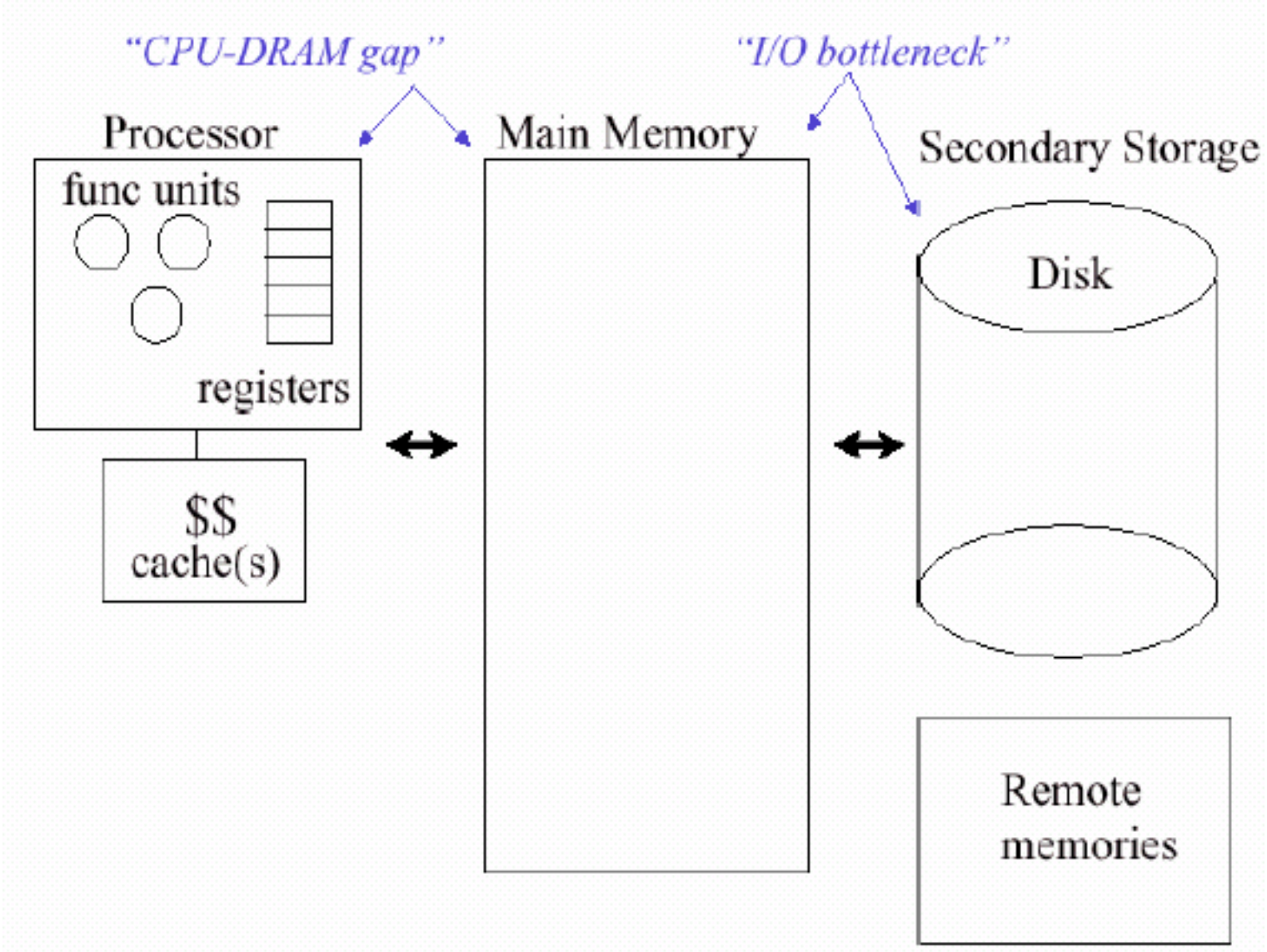
Probleme

- "Legarea" adreselor (*address binding*)
- Optimizări
  - încărcare dinamică
  - legare dinamică
  - *overlay*-uri
- Alocarea memoriei și fragmentarea ei

Întrebări

- Ce este *ierarhia de memorii* ?
- Ce este un *spațiu de adresare* ?
- Cum este *partajată* resursa memorie între toate procesele ce rulează în sistem?
- Cum poate un spațiu de adresare să fie *protejat* de operațiile executate de alte procese?
- Care este *overhead*-ul (i.e. încărcarea suplimentară) generat de operațiile cu memoria?

Ierarhii de memorii /1



Ierarhii de memorii /2

- Ierarhii de memorii:
  memorii cu acces mai rapid, dar realizate în tehnologie mai costisitoare și deci de dimensiune mai mică

  vs.

  memorii cu acces mai lent, dar realizate în tehnologie mai ieftină și deci de dimensiune mai mare
- Memorii: – internă: principală (RAM) ; cache (în CPU)
  – externă: secundară (discuri hard) ; de arhivare
- Astfel se realizează un mecanism, transparent pentru utilizator, prin care SC-ul lucrează cu o cantitate mică de memorie rapidă și una mare de memorie lentă, ca și cum ar avea numai memorie rapidă și în cantitate mare

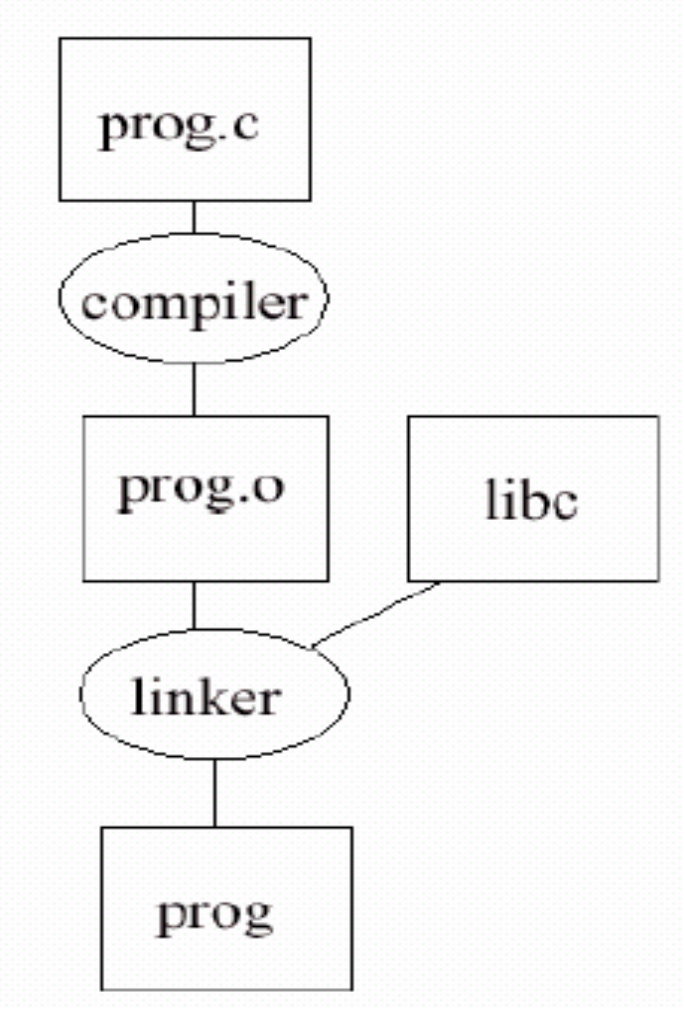
Legarea adreselor

- "Legarea" adreselor (*address binding*) : necesitatea de a decide unde să fie plasate codul executabil și datele
- Se poate face la:
  1. **momentul compilării** – se generează cod executabil cu adrese fixe. Programul trebuie recompilat dacă codul trebuie încărcat la o altă adresă de memorie (e.g. DOS fișiere .com)
  2. **momentul încărcării** – se generează adresele atunci când programul este încărcat în memorie (e.g. DOS fișiere .exe)
  3. **momentul execuției** (dinamic) – adresele sunt relative la o valoare care se poate modifica pe parcursul execuției, i.e. codul este relocabil la *runtime* (e.g. Windows fișiere .exe)

De la Program la Executabil

Fișierul executabil, rezultat prin compilarea codului sursă și link-editarea cu alte module compilate, conține:

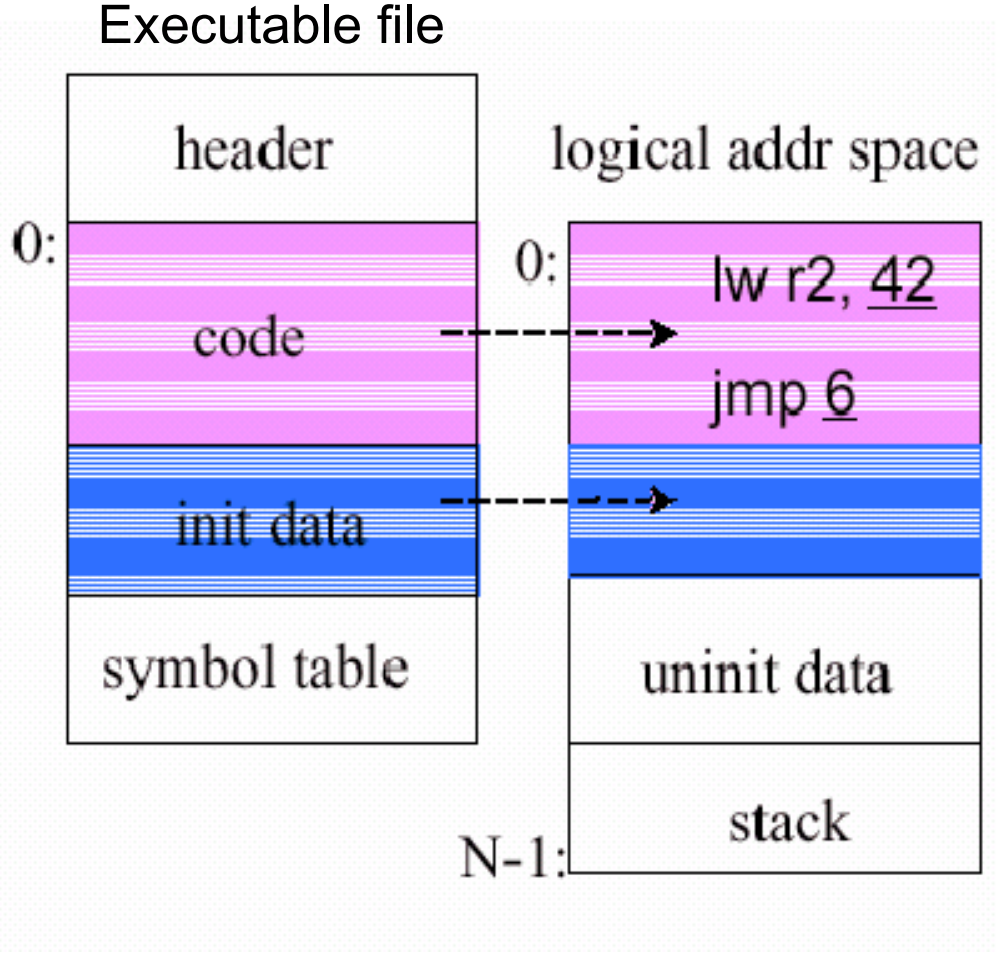
- instrucțiuni în limbaj mașina (ca și cum adresele încep de la zero)
- datele inițializate
- cât spațiu este necesar pentru datele neinițializate



De la Executabil la Spațiul de adresare

Atunci când este creat procesul, pe lângă codul și datele inițializate ce sunt copiate din fișierul executabil, mai trebuie rezervate adrese pentru zonele de date neinițializate și stivă.

Când și cum sunt asignate ***adresele fizice*** reale?



Optimizări /1

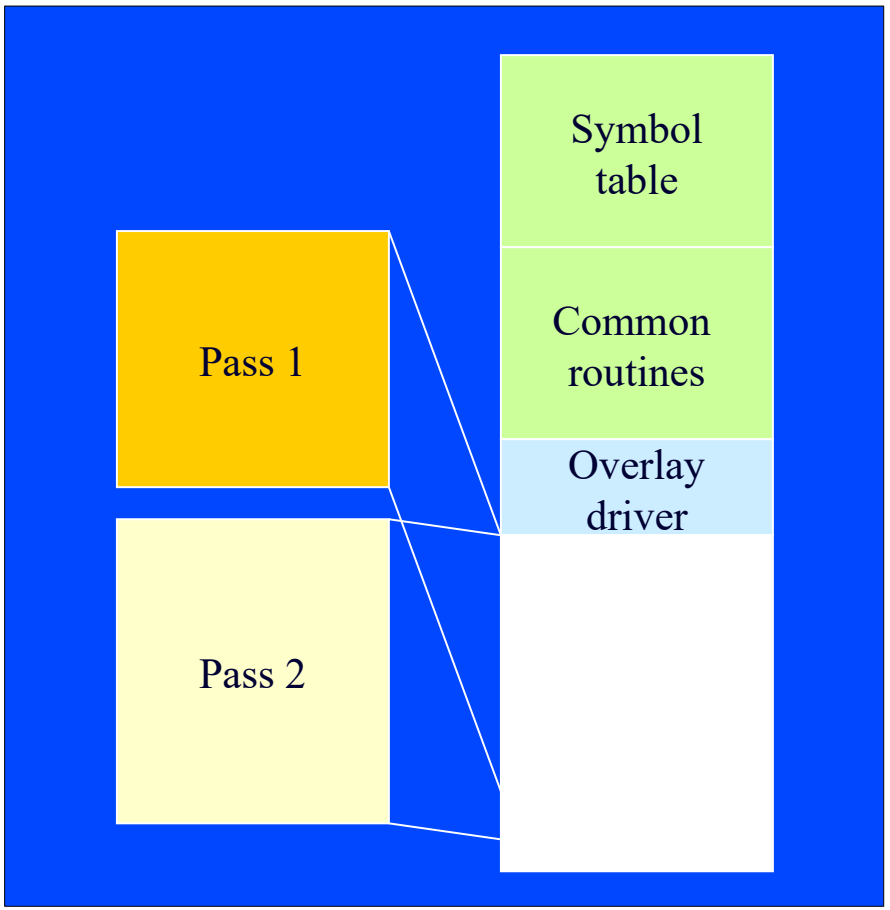
- **Încărcare dinamică**
  - Încărcarea rutinelor numai atunci când sunt necesare
  - O mai bună utilizare a memoriei deoarece codul utilizat rar nu este necesar să fie prezent în memorie pe toată durata execuției programului
  - Este implementată de programator prin intermediul unor apeluri de sistem ce încarcă noi "bucăți" de cod; complexitatea facilităților puse la dispoziție variază de la un SC la altul

Optimizări /2

- ***Overlay*-uri**
  - *Overlay*-urile furnizează un mod de scriere a programelor ce necesită mai multă memorie decât este fizic disponibilă (pe sisteme fără memorie virtuală)
  - Programatorul partiționează programul în "bucăți" de cod ce se pot suprapune; o "bucată" diferită de cea curentă este încărcată atunci când este nevoie de ea
  - Programarea este complexă și dificilă; programatorul este responsabil cu proiectarea "bucăților" de cod
  - ex.: DOS; suport din partea compilatorului Turbo Pascal

Optimizari /3

- ***Overlay*-uri** (cont.)
  - Exemplu: compilator multi-pass



Optimizări /4

- **Legarea dinamică (.DLL)** – Windows, OS/2
  - Legarea (*linking*-ul) subrutinelor este amânată până la momentul execuției programului
  - Rutinele din biblioteci (e.g. funcțiile C de bibliotecă) nu sunt incluse în codul obiect generat de compilator; în locul lor, este inclus un *stub* (ce conține informații despre cum poate fi localizată în memorie rutina respectivă ...)
  - Când este apelat, *stub*-ul se înlocuiește cu adresa rutinei respective și se execută aceasta
  - S.O.-ul trebuie să cunoască dacă rutina este prezentă în memorie, iar dacă nu este, trebuie să o încarce

Optimizări /5

- **Legarea dinamică (.DLL)** (cont.)
  - Avantaj: se permite *partajarea* codului
  - Probleme:
    – Programul depinde de versiunea .DLL-ului
    → .DLL hell
    (soluția: *assemblies* utilizate în .NET)
    – Programul nu va funcționa dacă .DLL-urile necesare nu sunt prezente în sistem
  - Echivalentul .DLL-urilor în UNIX/Linux: biblioteci partajate – bibliotecile **.so** (vezi /lib)

Alocarea memoriei /1

- Adrese de memorie logice vs. fizice
  - adrese logice (sau *virtuale*): adresele de accesare a memoriei generate de CPU (adresele din codul programului incarcat)
  - adrese fizice: adresele de accesare a hardware-ului memoriei (memoria fizică din SC)

- Pentru SC cu "legarea" adreselor la momentul compilării sau al încărcării, adresele logice coincid cu cele fizice
- Pentru SC cu "legarea" adreselor la momentul execuției, adresele logice nu mai coincid neapărat cu cele fizice

Translatarea adreselor logice în adrese fizice este executată de de către hardware-ul de mapare a memoriei (i.e. MMU); ea depinde de tipul acelui SC, în particular de mecanismul de administrare a memoriei utilizat de acel SC.

Alocarea memoriei /2

- Alocarea memoriei contiguă vs. necontiguă
  - alocare contiguă: pentru un proces (cod+date) se alocă o singură porțiune, continuă, din memoria fizică
  - alocare necontiguă: pentru un proces (cod+date) se alocă mai multe porțiuni separate din memoria fizică
- Memorie reală vs. virtuală
  - memoria reală: spațiul de adresare al proceselor este limitat de capacitatea memoriei interne
  - memoria virtuală: spațiul de adresare nu este limitat de capacitatea memoriei interne (aceasta este suplimentată de cea externă, prin tehnica de *swapping*)

Alocarea memoriei /3

- Scheme de alocare a memoriei:
  - alocarea contiguă:
    - alocarea unică, la S.O. mono-utilizator
    - alocarea cu partiții (la S.O. multi-utilizator)
      - cu partiții fixe (alocare statică)
      - cu partiții variabile (alocare dinamică)
    - alocarea cu *swapping*
  - alocarea necontiguă:
    - paginată (simplă și la cerere)
    - segmentată (simplă și la cerere)
    - segmentată-paginată (simplă și la cerere)

Scheme de alocare contiguă /1

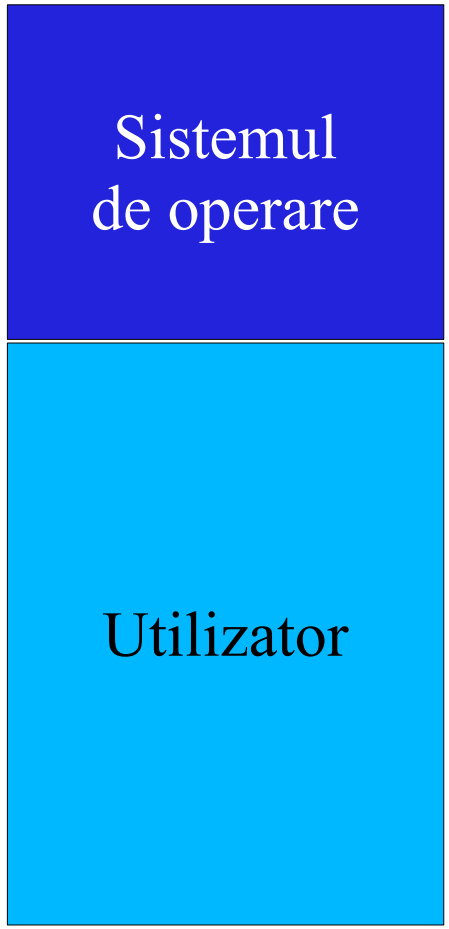
- **Alocarea unică (cu o singură partiție)**
  - **Nici o schemă de administrare a memoriei**
    - Prima formă de alocare d.p.d.v. istoric, folosită de primele SC, monoprogramate, fără S.O.
    - Furnizează utilizatorului o flexibilitate maximă
    - Fără servicii oferite de S.O. – fără control al întreruperilor, fără mecanisme de procesare a apelurilor de sistem și a erorilor, fără posibilități de multiprogramare

Scheme de alocare contiguă /2

- **Alocarea unică (cu două partiții)**
  - o partiție de memorie pentru utilizator
  - o partiție de memorie pentru S.O.
    - S.O.-ul este plasat fie în zona inferioară a memoriei (e.g. CP/M), fie în zona superioară a memoriei (e.g. DOS)
- Dezavantaje:

– neutilizarea optimă a memoriei și a CPU; imposibilitatea multiprogramării

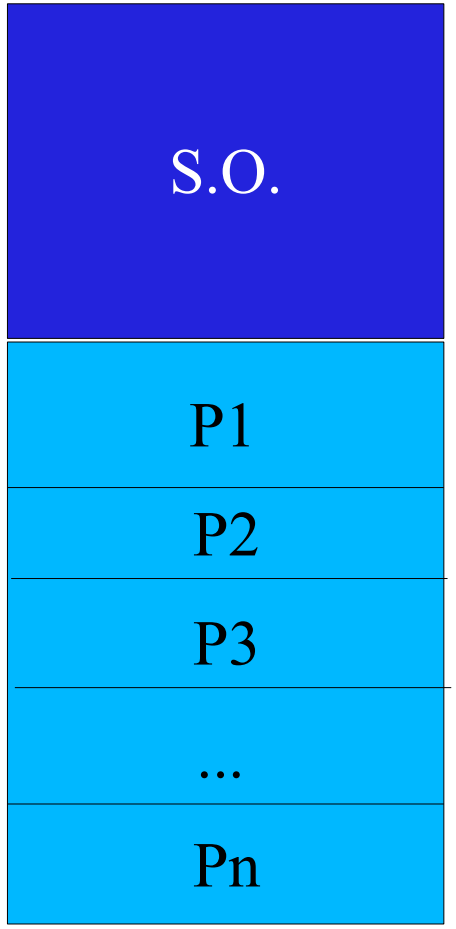
– programele s-ar putea să nu încapă în memoria disponibilă (excepție – programele cu *overlay*)



Scheme de alocare contiguă /3

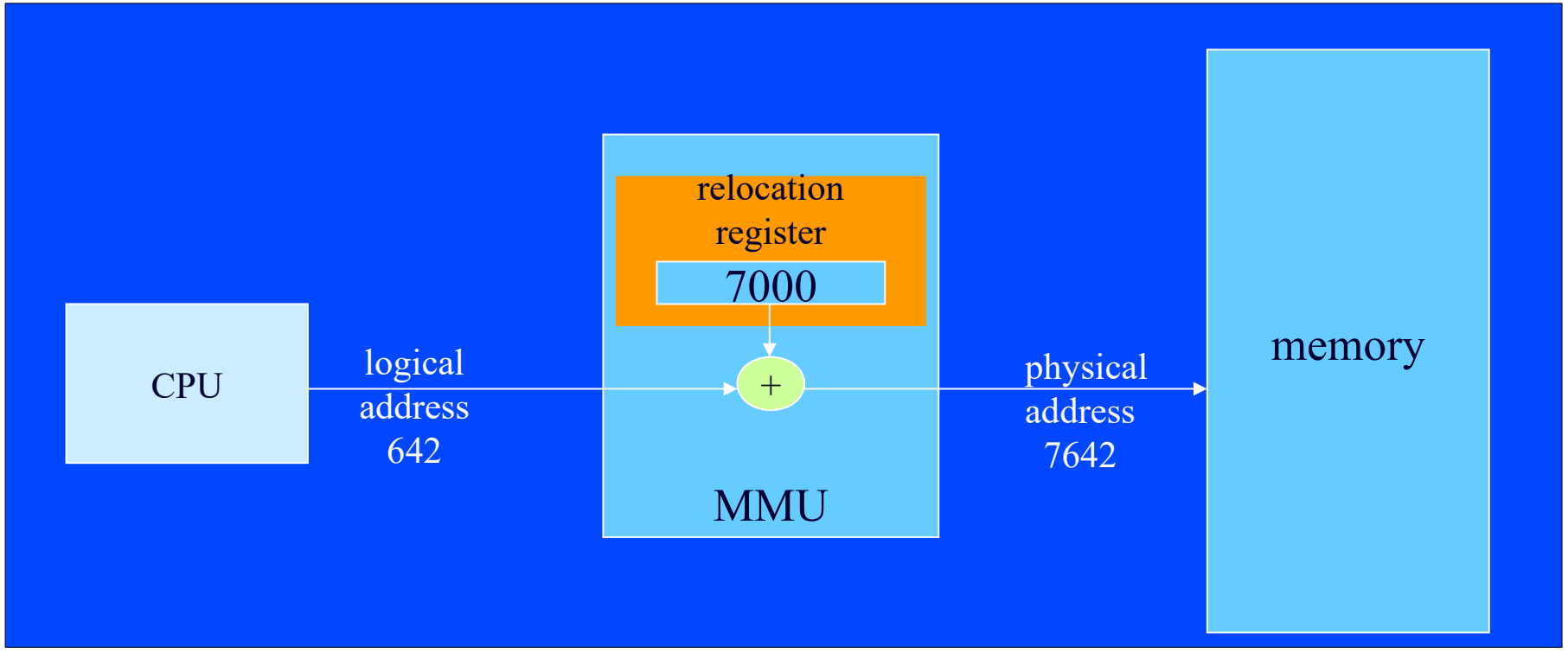
- **Alocarea memoriei în partiții fixe**
  - memoria este împărțită static în mai multe partiții (de dimensiuni nu neapărat egale)
  - fiecare partiție poate conține exact un proces (gradul de multiprogramare este limitat deci de numărul de partiții)
  - *alocarea absolută* – pentru SC cu legarea adreselor la compilare sau încărcare

  vs.
  - *alocarea relocabilă* – pentru SC cu legarea adreselor la execuție
  - utilizată la sistemele seriale (e.g. IBM OS/360)



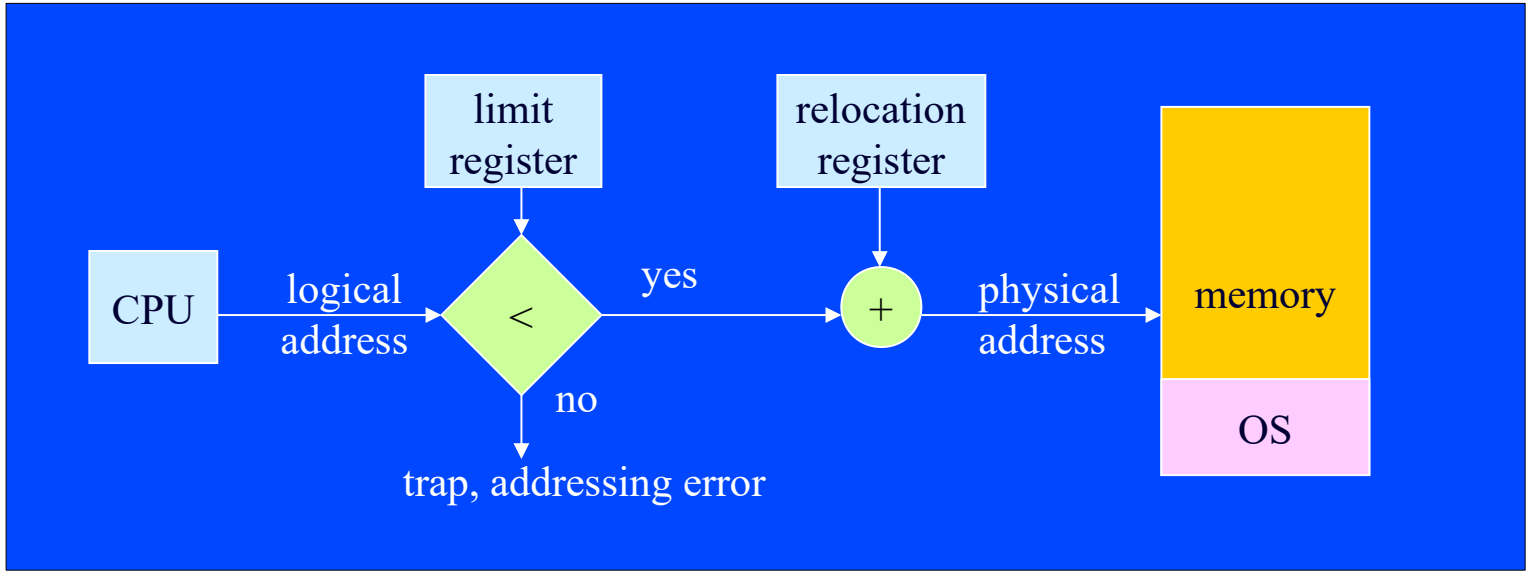
Scheme de alocare contiguă /4

- **Alocarea memoriei în partiții fixe** (cont.)
  - la alocarea relocabilă se folosește un **registru de relocare**
  - adresa fizică = adresa de bază (valoarea acestui registru) + deplasamentul (adresa logică)



Scheme de alocare contiguă /5

- **Alocarea memoriei în partiții fixe** (cont.)
  - S.O.-ul trebuie să asigure protecția partițiilor (proceselor)
  - un alt registru, **registrul de limită**, furnizează deplasamentul maxim ce se permite a se aduna la adresa de bază



  - un alt mecanism de protecție folosit de unele S.O.-uri: împărțirea partițiilor în pagini de memorie de lungime fixă (e.g. 2Ko) și folosirea de chei de acces la ele

Scheme de alocare contiguă /6

- **Alocarea memoriei în partiții fixe** (cont.)
  - când un proces se termină, se încarcă în partiția respectivă un alt proces dintre cele ce așteptau să le vină rândul la execuție
  - la primele SC seriale se folosea câte o coadă de așteptare pentru fiecare partiție, iar plasarea job-urilor pe partiții o realiza la început operatorul uman
  - apoi s-a folosit o singură coadă pentru toate partițiile, ceea ce a permis optimizarea plasării job-urilor (*job scheduler*)
  - Dezavantaje:
    - programele s-ar putea să nu încapă în partițiile disponibile
    - neutilizarea optimă a memoriei: poate rămâne spațiu nefolosit în fiecare partiție
    - fenomenul de fragmentare (internă) a memoriei

Scheme de alocare contiguă /7

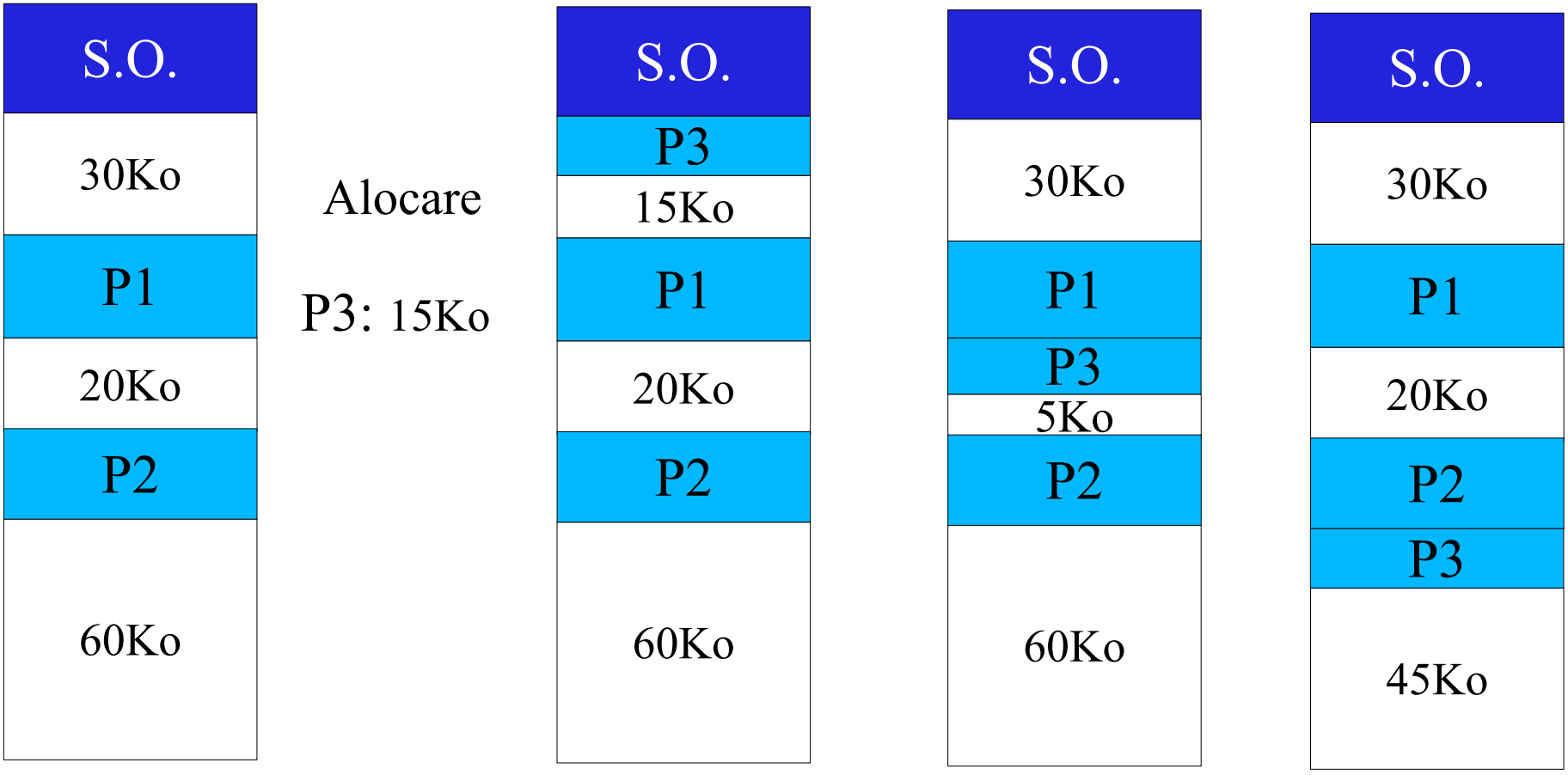
- **Alocarea dinamică (în partiții variabile)**
  - este tot o schemă de alocare contiguă
  - când se încarcă un proces în memorie, i se alocă exact spațiul de memorie necesar, din memoria *liberă* în acel moment, creându-se dinamic o partiție (de lungime variabilă)
  - când se termină un proces, partiția ocupată de el devine spațiu liber (și se unifică cu eventualul spațiu liber adiacent)
  - S.O.-ul trebuie să gestioneze o tabelă a partițiilor ocupate și o listă a spațiilor libere
  - S.O.-ul folosește un algoritm de alocare, ce decide ce spațiu liber să fie alocat unui proces la încărcarea sa
  - *alocarea absolută* vs. *alocarea relocabilă*
  - utilizată tot la sistemele seriale (prima dată la IBM OS/360)

Scheme de alocare contiguă /8

- **Alocarea dinamică (în partiții variabile)** (cont.)
  - algoritmi de alocare (de alegere a unui spațiu liber):
    - **FFA** (*first fit algorithm*): se parcurge lista spațiilor libere (ordonată crescător după adresa de început a spațiilor) și se alege *primul* spațiu de dimensiune suficientă (e.g. MINIX)
    - **BFA** (*best fit algorithm*): se parcurge lista spațiilor libere (ordonată crescător după dimensiunea spațiilor) și se alege *primul* spațiu de dimensiune suficientă (e.g. DOS)
      Produce cel mai mic spațiu liber rest.
    - **WFA** (*worst fit algorithm*): se parcurge lista spațiilor libere (ordonată crescător după dimensiunea spațiilor) și se alege *ultimul* spațiu din listă
      Produce cel mai mare spațiu liber rest.

Scheme de alocare contiguă /9

- **Alocarea dinamică (în partiții variabile)** (cont.)



a) FFA b) BFA c) WFA

Scheme de alocare contiguă /10

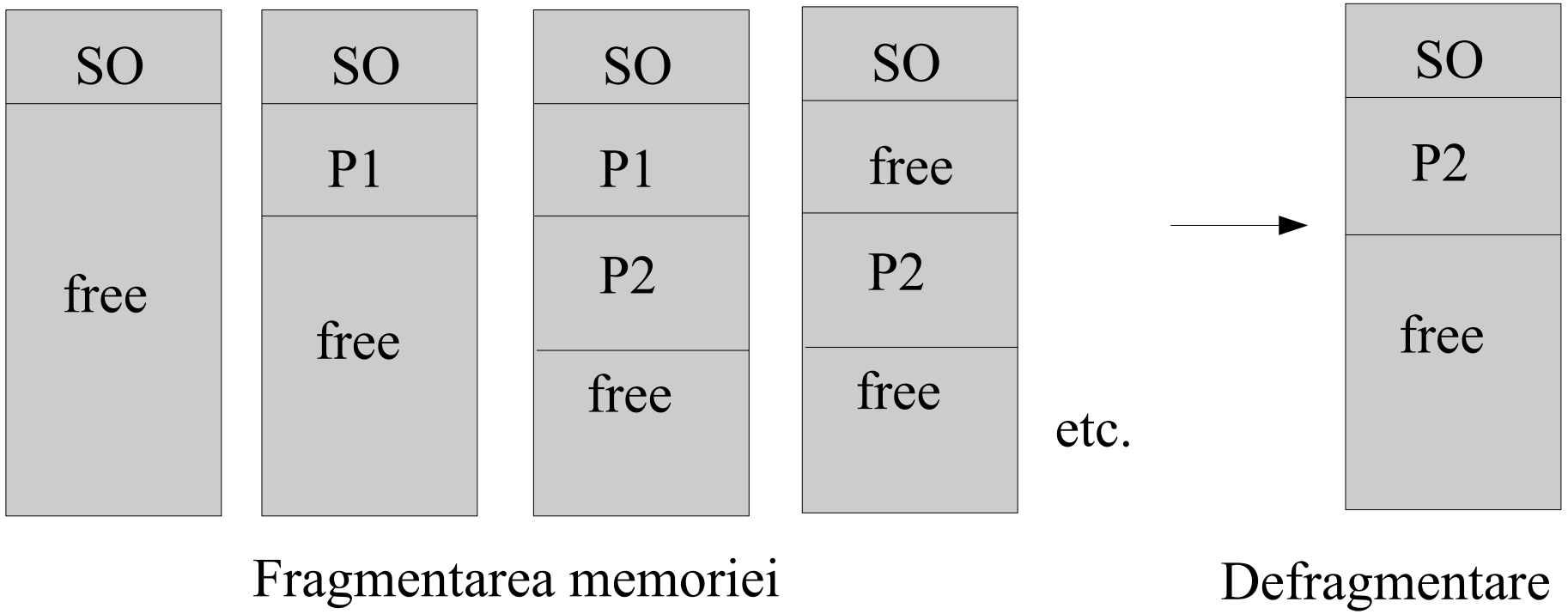
- **Alocarea dinamică (în partiții variabile)** (cont.)
  - în anumite scenarii oricare dintre cei 3 algoritmi este mai bun decât ceilalți; în general însă, FFA și BFA sunt mai buni decât WFA
  - problema: fenomenul de fragmentare a memoriei – în timp crește numărul spațiilor libere, iar dimensiunile acestora scad; se poate întâmpla ca un job să nu încapă în nici un spațiu liber, deși dimensiunea sa este mai mică decât suma spațiilor libere
  - Dezavantaje:
    - programele s-ar putea să nu încapă în spațiile libere
    - neutilizarea completă a memoriei: pot rămâne spații libere, nefolosite
    - fenomenul de fragmentare (externă) a memoriei

Scheme de alocare contiguă /11

- **Alocarea dinamică (în partiții variabile)** (cont.)
  - Ce se întâmplă când un proces nu are spațiu liber în care să se încarce? S.O.-ul poate lua următoarele decizii:
    - procesul așteaptă până când se va elibera o cantitate suficientă de memorie
    - efectuarea unei operații de *compactare* (*relocare*) a memoriei (i.e., o defragmentare a memoriei principale) – se deplasează partițiile ocupate pentru a se unifica (total sau parțial) spațiile libere
  - Compactarea rezolvă problema fragmentării memoriei, dar se poate aplica numai pentru *alocarea relocabilă* (i.e., când "legarea" adreselor este dinamică, la momentul execuției)

Scheme de alocare contigua /12

- **Alocarea dinamică (în partiții variabile)** (cont.)



Fragmentarea memoriei

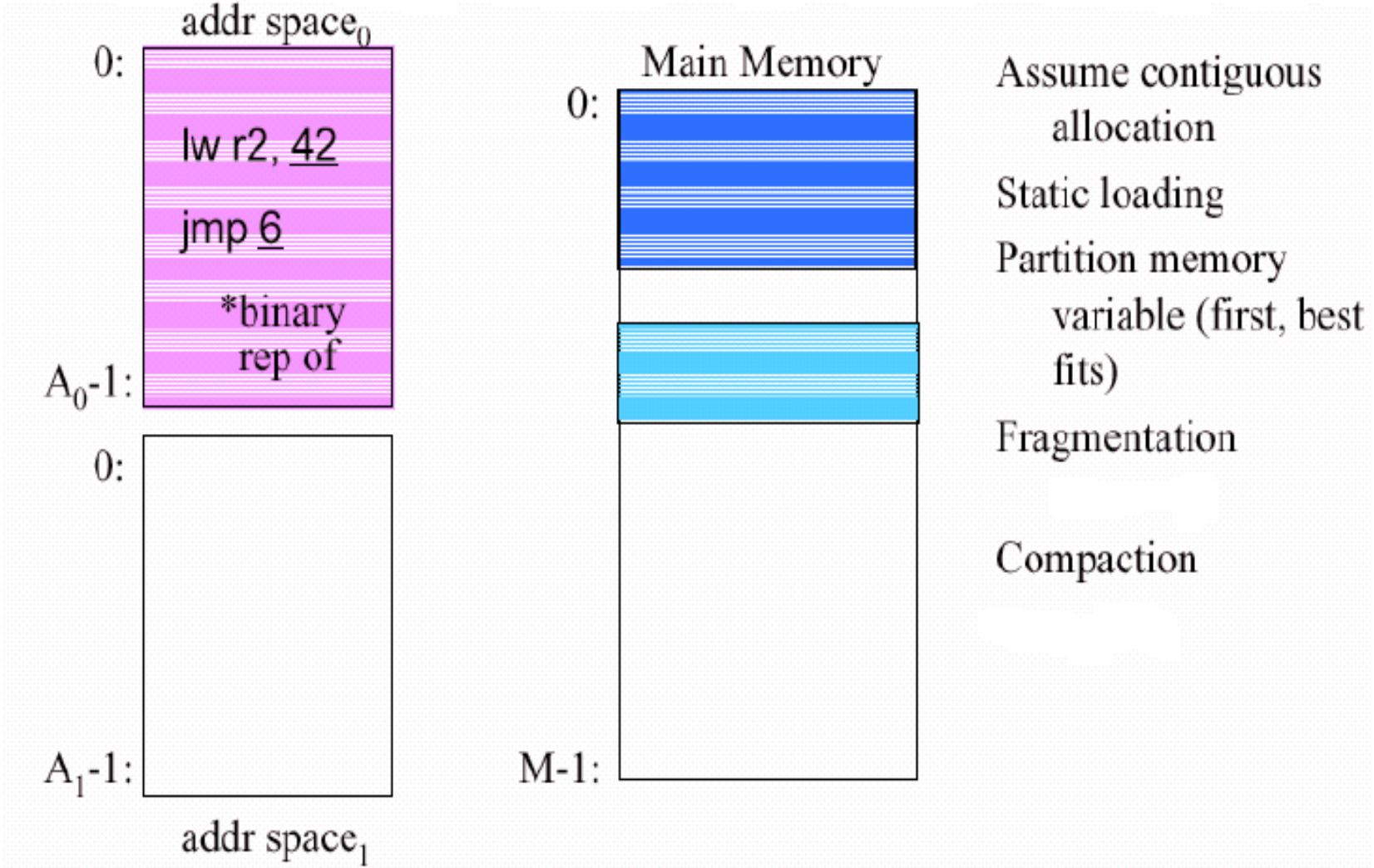
Defragmentare

Scheme de alocare contiguă /13

- **Alocarea dinamică (în partiții variabile)** (cont.)
  - Problema cu perifericele I/O: mecanismului DMA nu-i prieşte compactarea. Soluții: fie nu se permite compactarea în timpul operațiilor I/O, fie se folosesc buffere în spațiul de adrese al S.O. pentru operațiile I/O
  - Altă problemă: cum decid când să fac compactare?
  - Dezavantaj: *overhead*-ul implicat de compactare (i.e. de deplasarea unor zone voluminoase de memorie)
  - Între alocarea cu partiții fixe și cea cu partiții variabile nu există practic diferențe hardware; ambele sunt realizate prin intermediul unor rutine specializate, din cadrul S.O.-ului

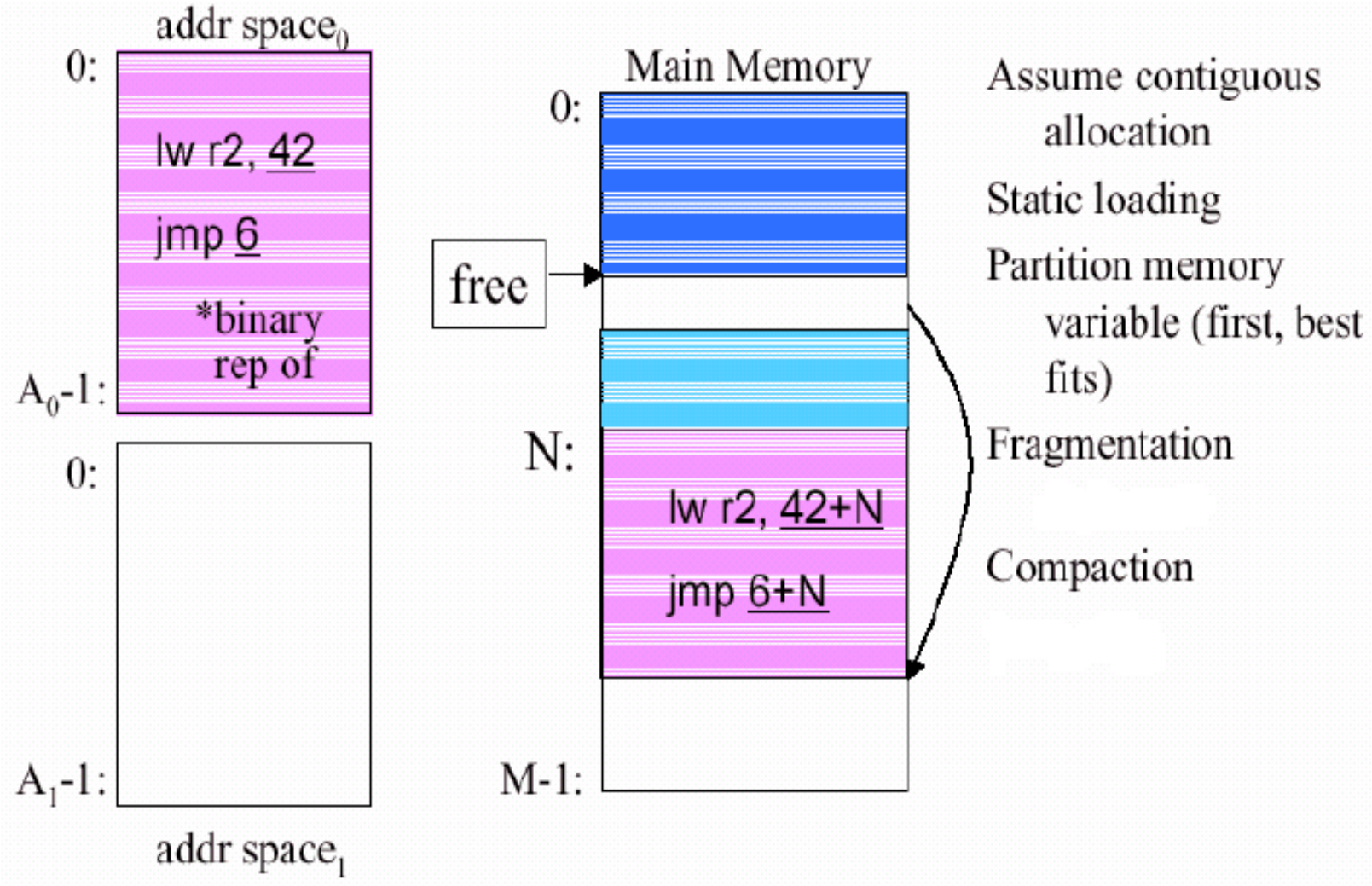
Scheme de alocare contiguă /14

- **Exemplu de alocare dinamică**



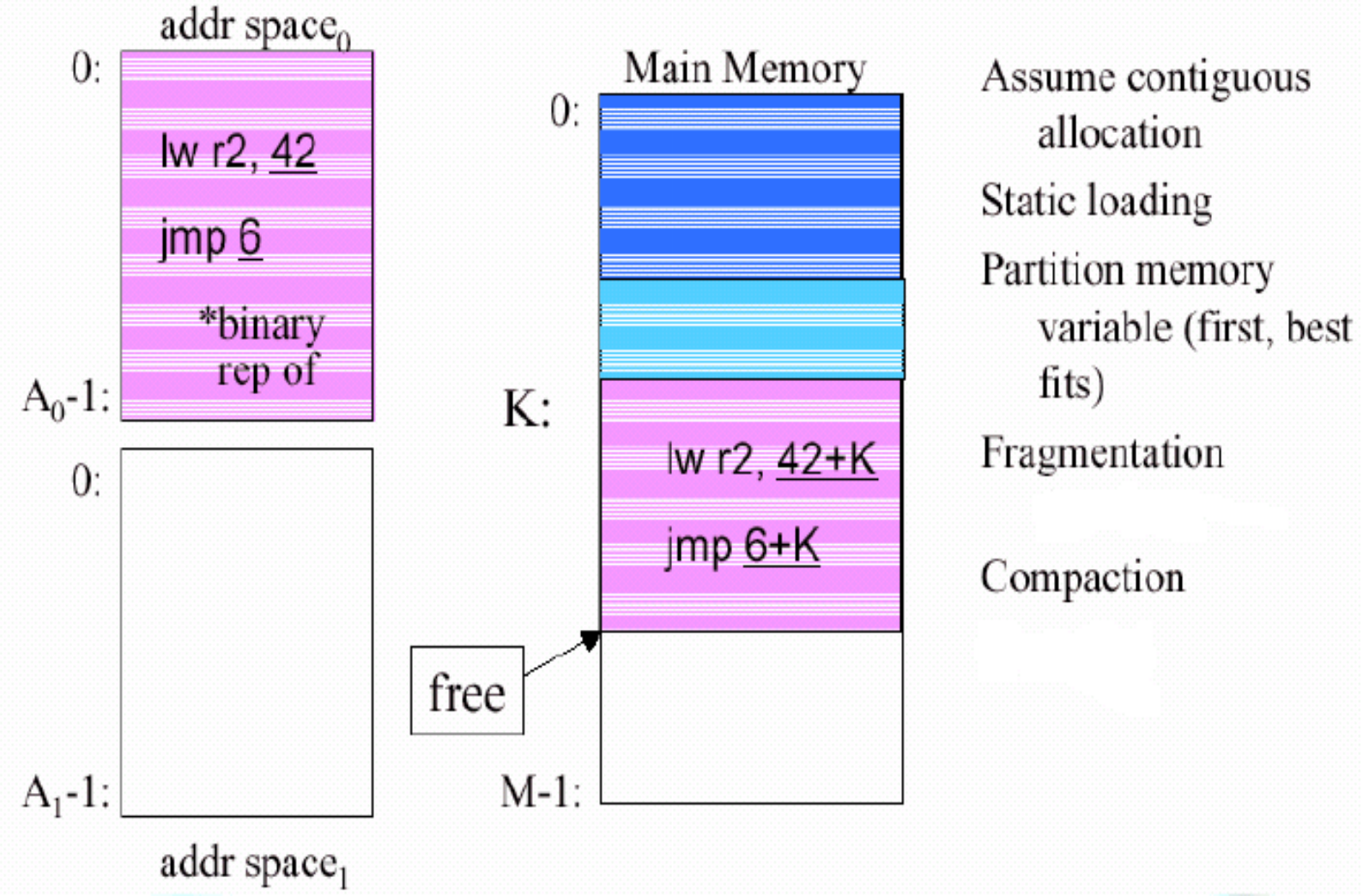
Scheme de alocare contiguă /15

- **Exemplu de alocare dinamică** (cont.)



Scheme de alocare contiguă /16

- **Exemplu de alocare dinamică** (cont.)



Scheme de alocare contiguă /17

Alocarea prin *swapping*

– după SC seriale au apărut SC cu *swapping*, denumite și SC *rollout-rollin*, sau SC cu *time-sharing*
– ideea: un proces aflat în starea de așteptare se evacuează temporar pe disc, eliberând astfel memoria principală ocupată
– reluarea execuției procesului se face prin reîncărcarea sa de pe disc în memorie
– e.g. S.O.-ul CTSS (Compatible Time-Sharing System)

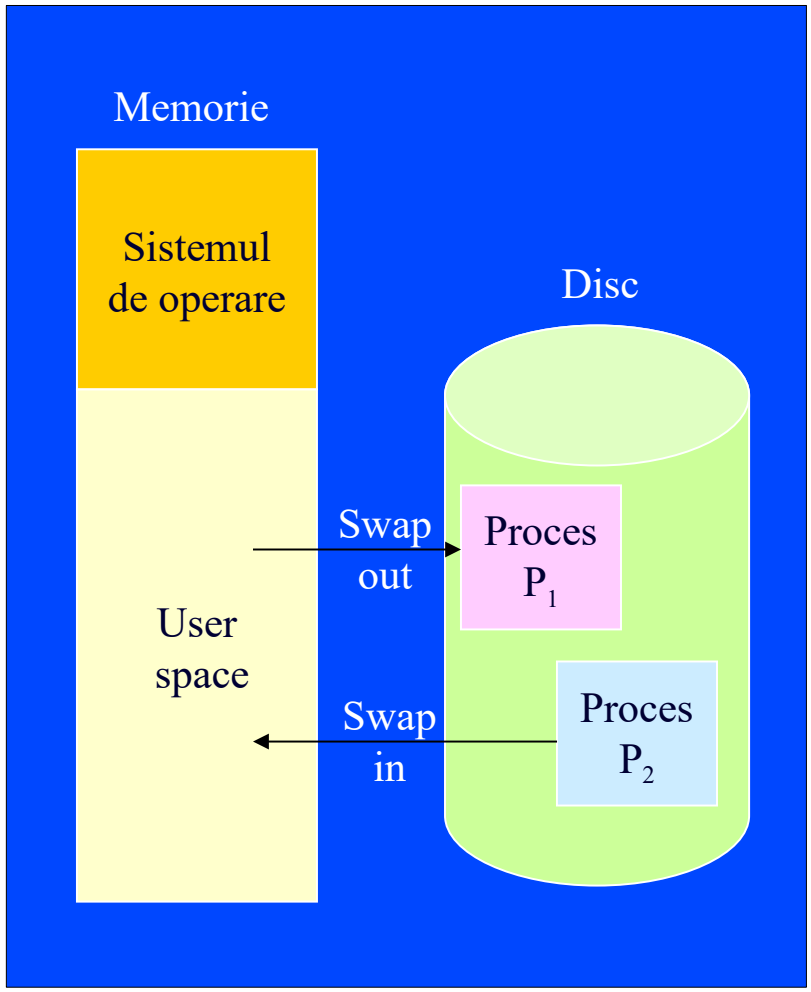
– Avantaje:
  • se simulează o memorie mai mare decât cea fizică existentă
  • se îmbunătățește folosirea resurselor SC (CPU + memorie)

Scheme de alocare contiguă /18

- **Alocarea prin *swapping*** (cont.)
  - SC seriale cu *time-sharing* foloseau pentru planificarea proceselor algoritmul RR; în acest caz evacuarea pe disc a jobului curent se face la expirarea cuantei sau la o cerere I/O, evacuare urmată de încărcarea altui proces de pe discul de swap în spațiul de memorie tocmai eliberat
  - SC seriale denumite *rollout-rollin* foloseau pentru planificarea proceselor algoritmul pe bază de priorități; în acest caz evacuarea pe disc se face când începe (sau redevine *ready*) un proces mai prioritar sau la o cerere I/O

Scheme de alocare contiguă /19

- **Alocarea prin *swapping***
  - Dezavantaje:
    1) transferul între memorie și disc este mare consumator de timp (deoarece viteza de acces a discului este cu câteva ordine de mărime mai mică decât cea a memoriei interne)
    2) alocarea este încă contiguă
    3) spațiul de adresare al unui proces nu poate depăși capacitatea memoriei interne



Memorie virtuală

Mecanisme de memorie virtuală

- SC cu *memorie virtuală* au capacitatea de a adresa un spațiu de memorie mai mare decât memoria internă disponibilă
- ideea: *swapping*-ul pe disc al unor "bucăți" de memorie, introdusă de S.O.-ul ATLAS (1960, Univ. Manchester, U.K.)
- tehnici de virtualizare: paginarea și segmentarea

- Alocarea prin *swapping* vs. Memoria virtuală
  - *swapping*: mutarea proceselor în întregime afară din memorie pe disc (și invers) atunci când este nevoie
  - memorie virtuală: mutarea unor mici "bucăți" de memorie afară din memorie pe disc (și invers) atunci când este nevoie

Bibliografie

- **Bibliografie obligatorie**

  capitolele despre *gestiunea memoriei* din

  – Silberschatz : "*Operating System Concepts*"

  (cap.9, prima parte: §9.1−2, din [OSC10])

  sau

  – Tanenbaum : "*Modern Operating Systems*"
  (cap.3, prima parte: §3.1−2, din [MOS4])

Sumar

- Introducere
- Probleme
- Ierarhii de memorie
- Alocarea memoriei
  - Adrese de memorie logice vs. fizice
  - Scheme de alocare contigue
  - Memorie virtuală

  (va urma)
  - Scheme de alocare necontigue

Întrebări ?