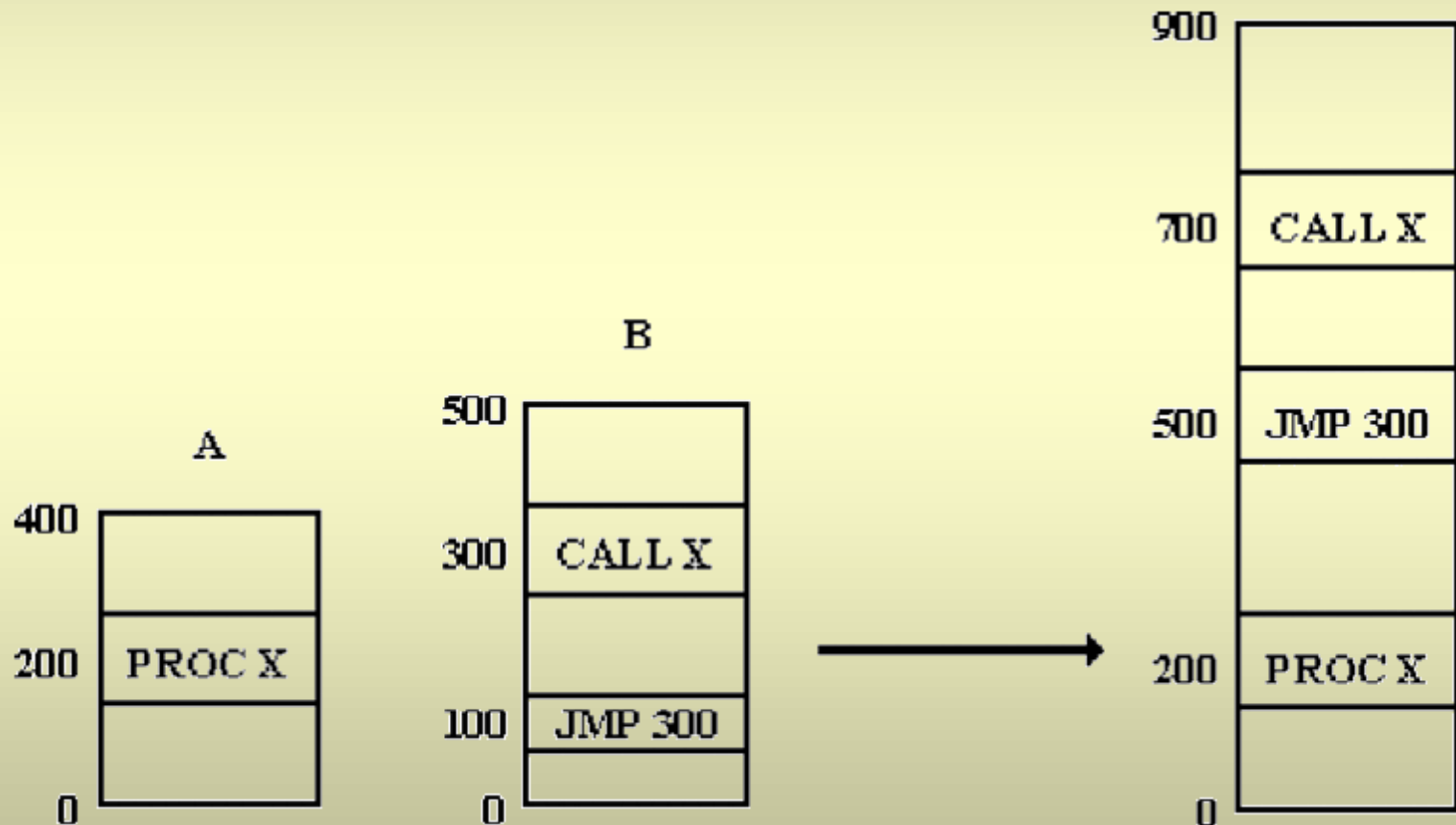


## V.6. Crearea și execuția programelor

# Crearea unui program - faze

- compilarea
  - traducerea comenzilor scrise într-un limbaj sursă în instrucțiuni pentru procesor
- editarea legăturilor (*linking*)
  - tratează aspecte privitoare la gestiunea memoriei într-un program

# Crearea unui fișier executabil din mai multe module sursă



## Problema relocării

- instrucțiunea de salt - adresa de salt nu mai este corectă
- module compilate independent - fiecare presupune că începe la adresa 0
- afectează și instrucțiunile care accesează date (adrese de memorie)
- adresele sunt relocate (deplasate) față de momentul compilării

# Problema referințelor externe

- funcția X - apelată din alt modul decât cel în care este definită
- la momentul compilării
  - se știe că este definită în alt modul
  - este imposibil de determinat la ce adresă se va găsi funcția în programul final

# Crearea programelor

- se poate scrie un program dintr-un singur modul?
- nu întotdeauna
- programe foarte complexe - modularitate
- biblioteci de funcții - module separate
  - precompilate
  - codul sursă nu este disponibil

# Fazele creării unui program

- compilarea modulelor
  - fișier sursă → fișier obiect
  - fișierele obiect conțin informații necesare în faza editării de legături
- editarea legăturilor
  - fișiere obiect → fișier executabil
  - se folosesc informațiile din fișierele obiect

# Structura unui fișier obiect (1)

## 1. antetul

- informații de identificare
- informații despre celelalte părți ale fișierului

## 2. tabela punctelor de intrare

- conține numele simbolurilor (variabile și funcții) din modulul curent care pot fi apelate din alte module



## Structura unui fișier obiect (2)

### 3. tabela referințelor externe

- conține numele simbolurilor definite în alte module, dar utilizate în modulul curent

### 4. codul propriu-zis

- rezultat din compilare
- singura parte care va apărea în fișierul executabil

# Structura unui fișier obiect (3)

## 5. dicționarul de relocare

- conține informații despre localizarea instrucțiunilor din partea de cod care necesită modificarea adreselor cu care lucrează
- forme de memorare
  - hartă de biți
  - listă înlănțuită

## Editorul de legături (1)

1. construiește o tabelă cu toate modulele obiect și dimensiunile acestora
2. pe baza acestei tabele atribuie adrese de start modulelor obiect
  - adresa de start a unui modul = suma dimensiunilor modulelor anterioare

## Editorul de legături (2)

3. determină instrucțiunile care realizează accese la memorie și adună la fiecare adresă o constantă de relocare
  - egală cu adresa de start a modulului din care face parte
4. determină instrucțiunile care apelează funcții sau date din alte module și inserează adresele corespunzătoare

# Execuția programelor

- la ce adresă începe programul când este încărcat în memorie?
- nu se știe la momentul când este creat
- toate adresele din program depind de adresa de început
- concluzie: problema relocării apare din nou la lansarea programului în execuție

## Soluția 1

- Fișierul executabil conține informații de relocare
  - aceste informații sunt utilizate de sistemul de operare la încărcarea programului în memorie
  - pentru a actualiza referințele la memorie
  - exemplu: sistemul de operare DOS

## Soluția 2

- Utilizarea unui registru de relocare
  - încărcat întotdeauna cu valoarea adresei de început a programului curent
  - acces la memorie - la adresa precizată prin instrucțiune se adună valoarea din registrul de relocare
  - dependentă de hardware
    - nu toate procesoarele au registru de relocare

## Soluția 3

- Programele conțin numai referiri la memorie relative la contorul program
  - program independent de poziție
  - poate fi încărcat în memorie la orice adresă
  - foarte greu de scris
    - instrucțiuni de salt relative - cu restricții
    - instrucțiuni care lucrează cu adrese de date relative la contorul program - nu există



## Soluția 4

- Paginarea memoriei
  - programul poate fi mutat oriunde în memoria fizică
  - programul crede că începe de la adresa 0, chiar dacă nu este așa
  - dependentă de suportul hardware (mecanismul de paginare)

# Biblioteci partajate (1)

## Legare dinamică

- proceduri și variabile care nu sunt incluse permanent în program
  - numai atunci când este nevoie de ele
- proceduri și variabile partajate de mai multe programe

## Biblioteci partajate (2)

### Utilitatea legării dinamice

- proceduri care tratează situații excepționale
  - rar apelate
  - ar ocupa inutil memoria
- proceduri folosite de multe programe
  - o singură copie pe disc
  - o singură instanță încărcată în memorie

# Biblioteci partajate (3)

Tipuri de legare dinamică

- implicită
- explicită

# Legare implicită

- folosește biblioteci de import
  - legate static în fișierul executabil
  - indică bibliotecile partajate necesare programului
- la lansarea programului
  - sistemul de operare verifică bibliotecile de import
  - încarcă în memorie bibliotecile partajate care lipsesc

## Legare explicită (1)

- programul face un apel sistem specific
- cere legarea unei anumite biblioteci partajate
- dacă biblioteca nu există deja în memorie, este încărcată
- legătura cu o bibliotecă partajată poate fi realizată sau distrusă în orice moment

## Legare explicită (2)

- exemplu - Windows

```
//legare explicită a unui modul  
hLib=LoadLibrary("module");  
//se obține un pointer la o funcție  
fAddr=GetProcAddress(hLib,"func");  
(fAddr)(2,3,8); //apel funcție  
FreeLibrary(hLib); //eliberare modul  
(fAddr)(2,3,8); //eroare, funcția nu  
mai este disponibilă
```

## Legare explicită (3)

- **exemplu - Linux**

```
//legare explicită a unui modul  
hLib=dlopen("module",RTLD_LAZY);  
//se obține un pointer la o funcție  
fAddr=dlsym(hLib,"func");  
(fAddr)(2,3,8); //apel funcție  
dlclose(hLib); //eliberare modul  
(fAddr)(2,3,8); //eroare, funcția nu  
mai este disponibilă
```