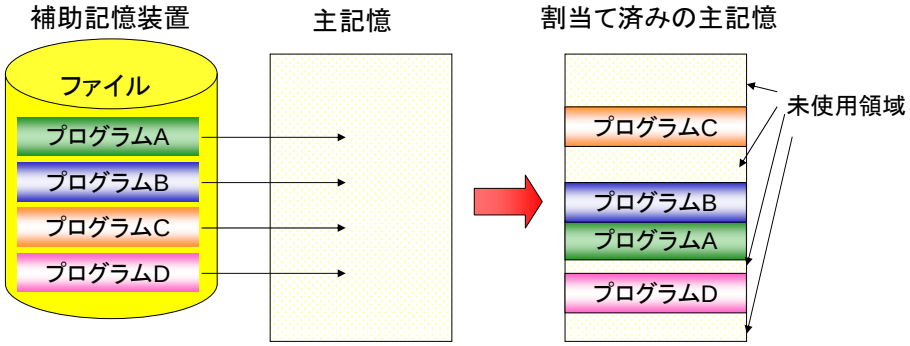
主記憶管理(メモリ管理)

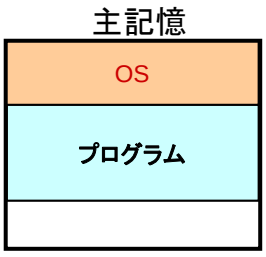
- プログラムは主記憶上で実行される
- プログラムやデータはファイルとして補助記憶装置に格納されている
- 複数のプログラムを同時に主記憶上で実行する(マルチプロセス)

OSが管理すること:主記憶の効率的に使う



主記憶の割当て方式

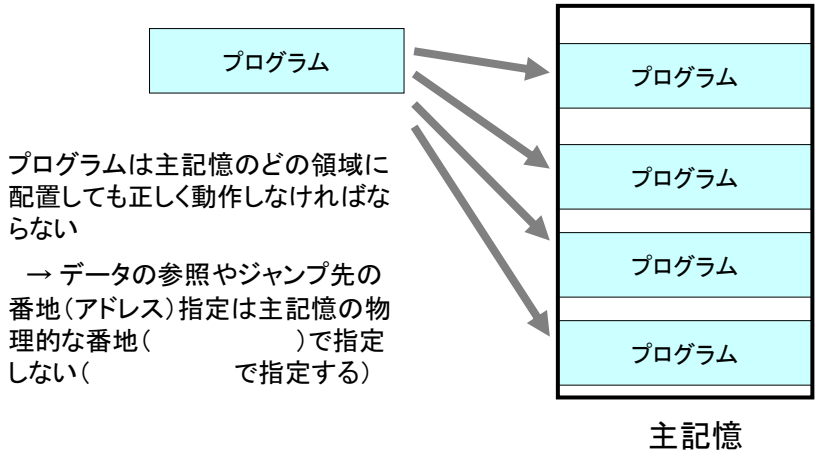
- 単一プロセスにおける主記憶割当て
  - 単一連続割当て
    : OSと一つのユーザプログラムが主記憶領域を分割して使用する
    : ユーザプログラムはOSが使用している領域以外を使用する



- マルチプロセスにおける主記憶割当て
  - 固定区画方式
  - 可変区画方式

→ (1) (2) プログラムに要求される性質
(3) アドレス変換 — ハードウェアによる支援

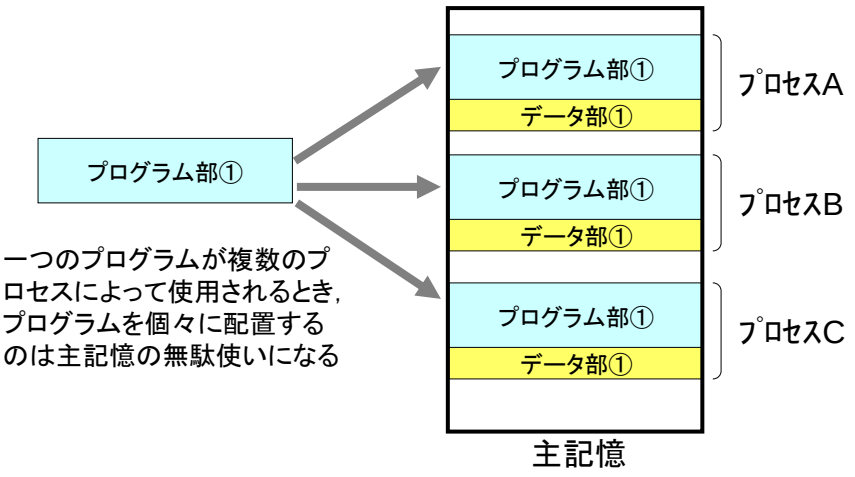
(1) 再配置可能（relocatable）であるということ



プログラムは主記憶のどの領域に配置しても正しく動作しなければならない

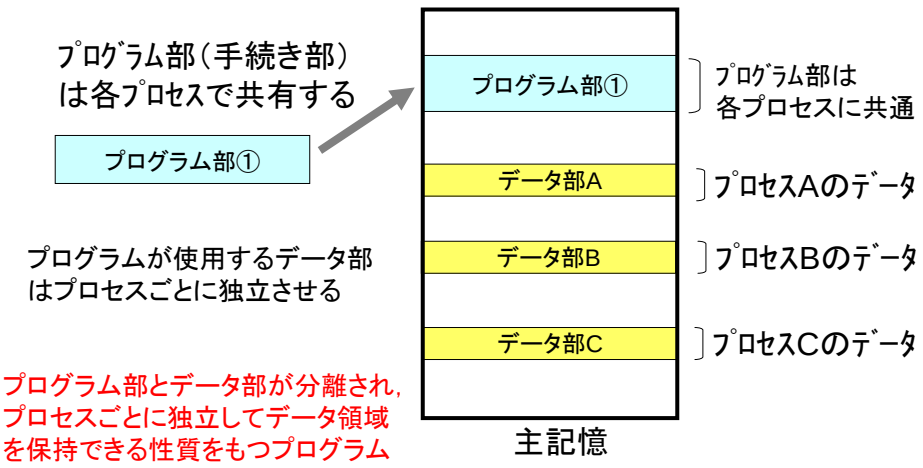
→ データの参照やジャンプ先の番地（アドレス）指定は主記憶の物理的な番地（　　）で指定しない（　　で指定する）

(2) 再入可能（reentrant）であるということ



一つのプログラムが複数のプロセスによって使用されるとき，プログラムを個々に配置するのは主記憶の無駄使いになる

(2) 再入可能(reentrant)であるということ



プログラム部とデータ部が分離され，
プロセスごとに独立してデータ領域
を保持できる性質をもつプログラム

＝ 再入可能プログラム / リエントラントプログラム

ソフトウェア開発技術者試験問題(平成16年度春期)

問:再入可能(リエントラント)プログラムの説明として，最も適切なものはどれか。

ア 一度実行した後，ロードし直さずに再び実行を繰り返しても，正しい結果が得られる。

イ 実記憶上のどこのアドレスに配置しても実行することが可能である。

ウ 複数のセグメントに分割されており，セグメント単位にロードして実行することが可能である。

エ 複数のプロセスで並行して実行しても，正しい結果が得られる。

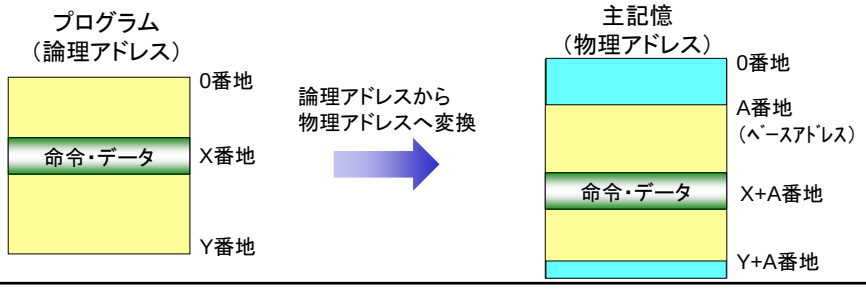
基本情報技術者試験問題（平成17年度春期）

問：処理が終了していないプログラムが，別のプログラムから再度呼出されることがある。このプログラムが正しく実行されるために備えるべき性質はどれか。

ア　再帰的（リカーシブ）
イ　再使用可能（リユーザブル）
ウ　再入可能（リエントラント）
エ　再配置可能（リロケータブル）

物理アドレスと論理アドレス

- マルチプロセスの環境では，OSがプログラムの主記憶領域中の配置場所を決める
  → プログラム自身は実行時の物理アドレスを意識できない
  - ◆ プログラム実行時の命令やデータの主記憶上の番地（アドレス）：
  - ◆ プログラムが想定する命令やデータの論理的な番地（アドレス）：
- プログラムの実行時には，「論理アドレス」は「物理アドレス」に変換されて実行される



(3) アドレス変換（address mapping/translation）

- 静的再配置
  - 補助記憶装置から主記憶へプログラムをもってくる時（ロード時）に物理アドレスに変換する
  - 実現が容易
- 動的再配置
  - 命令の実行時にハードウェア（ベースレジスタ）によって論理アドレスを物理アドレスへ変換する（後述）
  - コンパクション（後述）によって、主記憶の利用効率を向上できる

主記憶の割当て方式

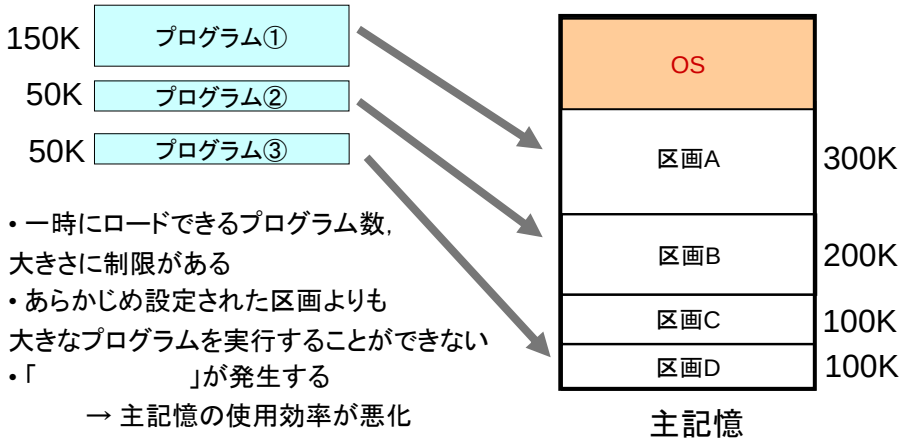
- 単一プロセスにおける主記憶割当て
  - 単一連続割当て
    - ：OSと一つのユーザプログラムが主記憶領域を分割して使用する
    - ：ユーザプログラムはOSが使用している領域以外を使用する

- マルチプロセスにおける主記憶割当て
  - 固定区画方式
  - 可変区画方式

→ (1) 再配置可能
(2) 再入可能
（プログラムに要求される性質）
(3) アドレス変換 — ハードウェアのよる支援

固定区画方式

- 主記憶領域をサイズ固定の区画に分割
- 領域サイズを複数用意する
- 区画内に収まるプログラムを選んでロードする

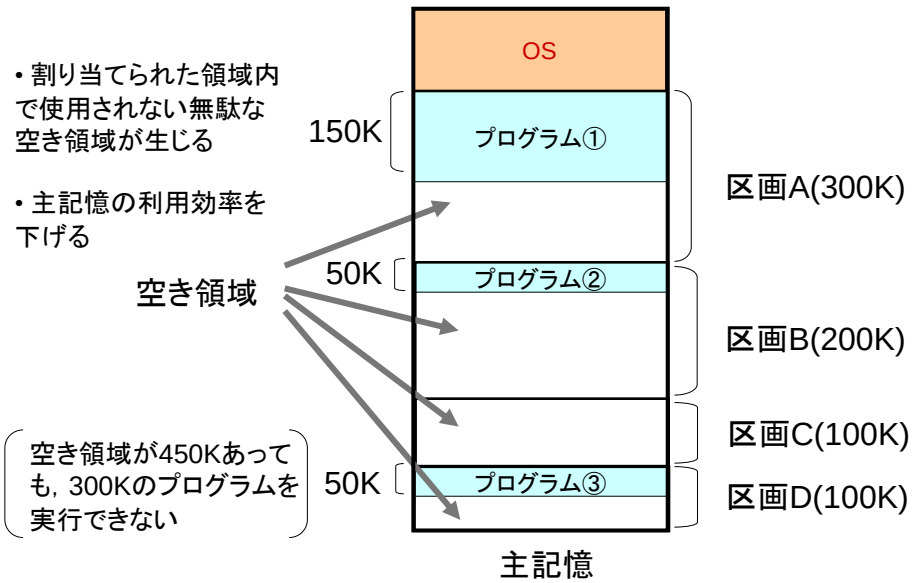


• 一時にロードできるプログラム数，大きさに制限がある
• あらかじめ設定された区画よりも大きなプログラムを実行することができない
• 「　　」が発生する
→ 主記憶の使用効率が悪化

内部断片化

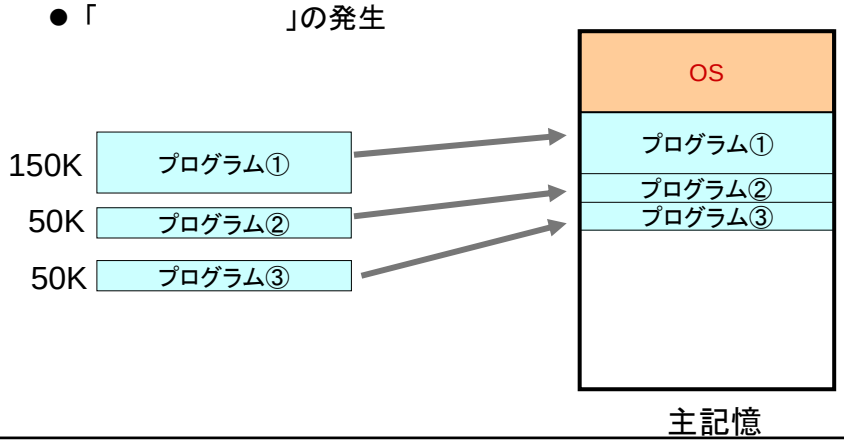
• 割り当てられた領域内で使用されない無駄な空き領域が生じる

• 主記憶の利用効率を下げる



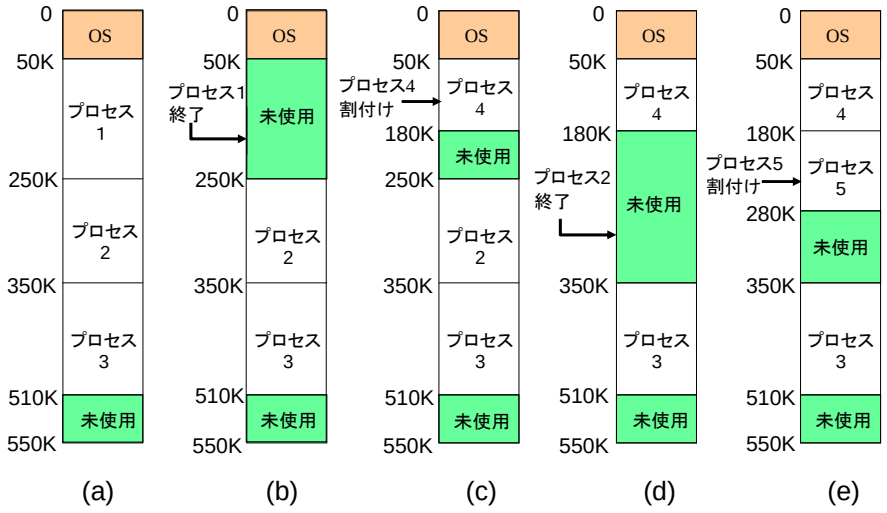
可変区画方式

- メモリ領域を可変長の区画で管理する
- 主記憶に格納されるプログラム数，大きさは動的に定まる
- 新しいプログラムをロードするとき，そのプログラムを格納するのに十分な空き領域を確保し，その領域に割り当てる
- 「　　」の発生



可変区画割当ての例

（P1：200K　P2：100K　P3：160K　P4：130K　P5：100K）

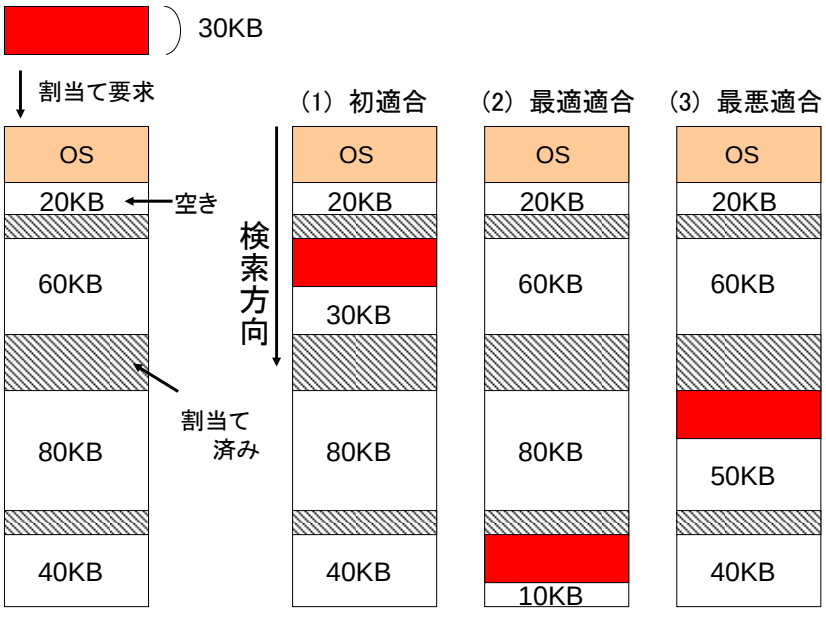


可変区画方式における空き領域の割当てアルゴリズム

◆ 新しいプログラムをロードするための主記憶中に散在する空き領域を探す

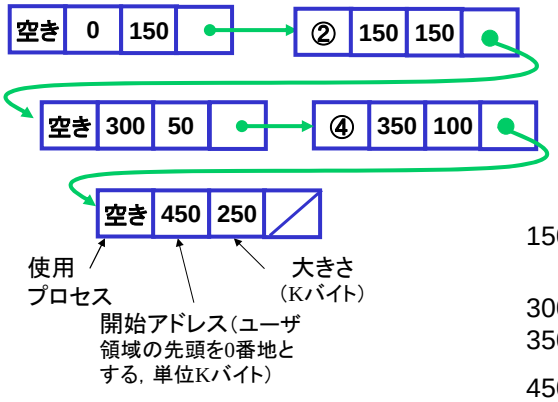
1. 初適合（first-fit）... 最初に見つかった十分な空き領域を割り当てる
   - 探索が速く、実現が容易、利用効率はあまりよくない
2. 最適適合(best-fit) ... 十分な領域をもつ最小の空き領域を割り当てる
   - 利用効率はよい（空き領域は最小）が、空き領域を見つけるオーバーヘッドが大きい
3. 最悪適合（worst-fit） ... 最大の空き領域を割り当てる
   - 小さな空き領域が多くなり，領域の利用効率が悪い

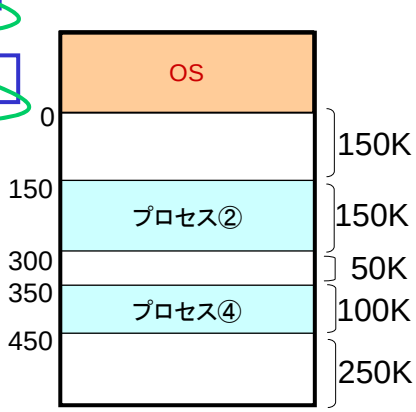
メモリ領域割当てアルゴリズム



空き領域の管理方法(1)

1. リスト方式





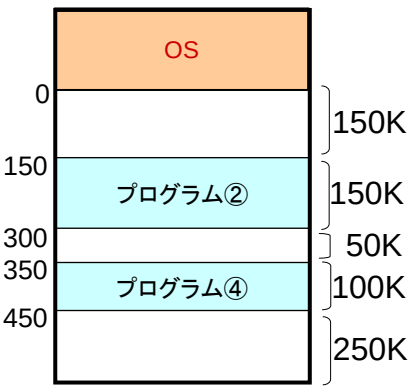
主記憶

空き領域の管理方法(2)

2. ビットマップ方式

0 0 0 0 0 0 1 1
1 1 1 1 0 0 1 1
1 1 0 0 0 0 0 0
0 0 0 0

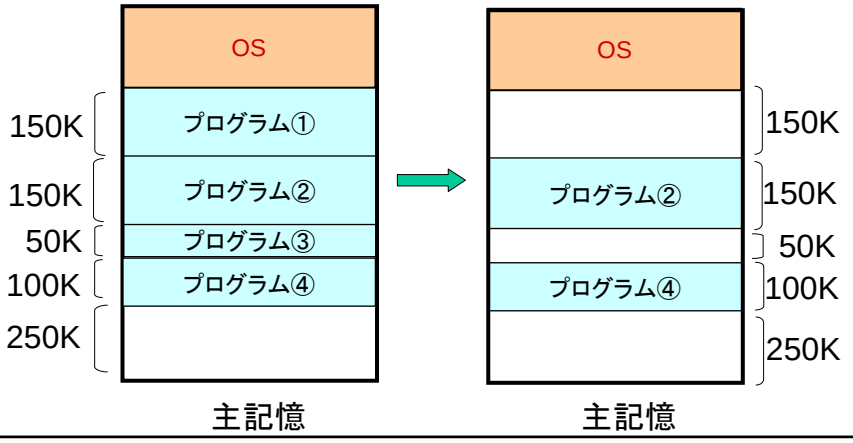
領域の未使用/使用を0と1に対応させる（この例では25Kバイトを1ビットに対応させている）



主記憶

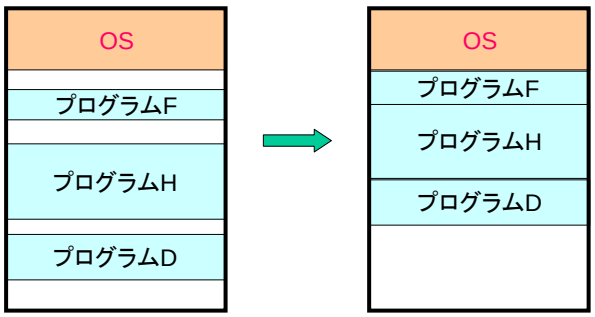
外部断片化

- ロードされたプログラムの実行が終了し，空き領域が断片的に存在する。
- 全体としての空き領域が大きくても，一つ一つの空き領域が小さいため、新たなプログラムを割り当てられない。



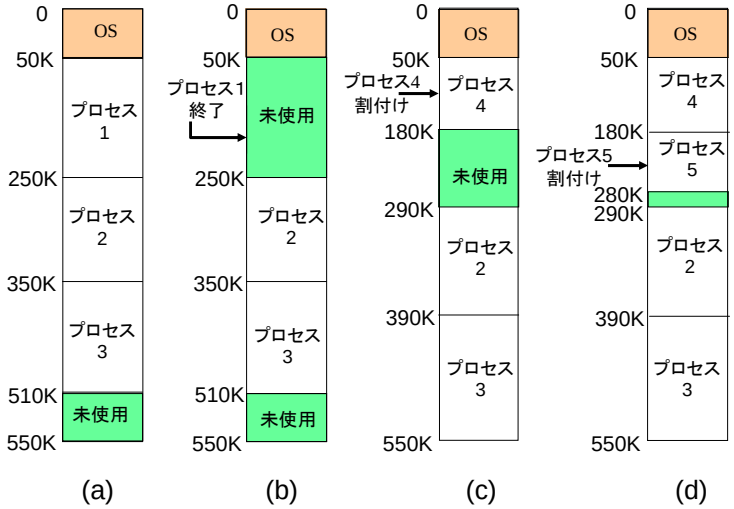
外部断片化の解消：コンパクション

- 主記憶上のプログラムを移動することによって、散在する空き領域を1つの大きい空き領域にまとめる
  → 主記憶の利用効率が向上する
- プログラムの移動によるオーバーヘッドが大きい
- プログラムは　　可能でなければならない



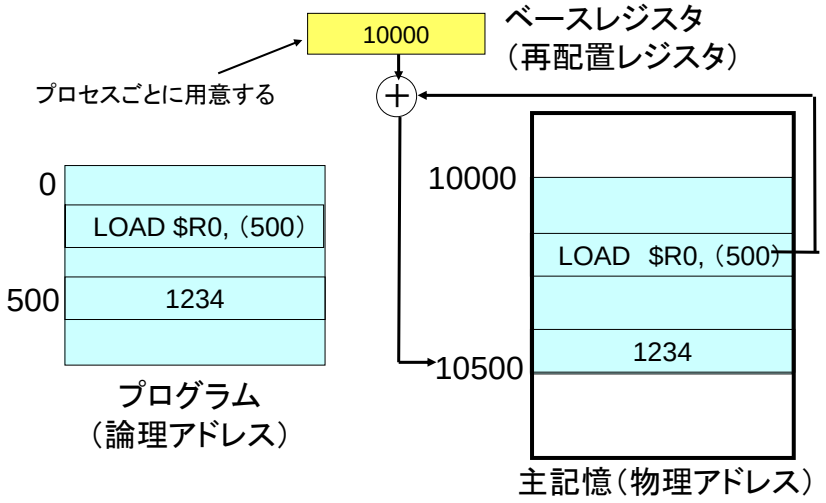
コンパクションを行った場合の可変区画割当ての例

（P1：200K　P2：100K　P3：160K　P4：130K　P5：100K）



動的再配置（dynamic relocation)

–



演習問題 5

1. 次の文に相当する方式・状況・現象を何というか。選択群から選べ。

(1) 複数のプロセスが互いに干渉することなく，同一のプログラムを同時に利用できること。
(2) 主記憶のどこに読み込まれても実行可能であること
(3) 命令が，命令やデータの記憶場所を識別するのに使用する論理的なアドレス
(4) 命令やデータの実体が存在する主記憶上の物理的なアドレス
(5) プログラムの主記憶へのロード時に，論理アドレスを物理アドレスに変換して，主記憶に配置する方式
(6) プログラムの実行時に，ハードウェアによって論理アドレスを物理アドレスに変換してプログラムを実行する方式
(7) 割当てられた主記憶領域の中で無駄な領域が生じる現象
(8) 主記憶の空き領域が全体として大きくても，それらが主記憶内に散在しているため，プログラムの実行に利用することができない現象

【選択群】

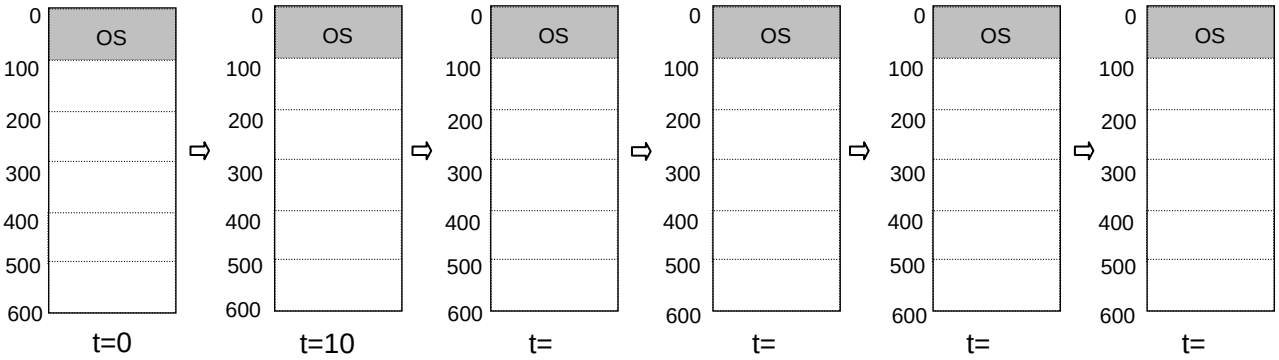
内部断片化	**物理アドレス**	**外部断片化**	**動的再配置**
再入可能	**再配置可能**	**論理アドレス**	**静的再配置**

2. 右表の 5 個のプロセスが順に主記憶にロードされて実行されるときのメモリ内容の推移を(a),(b)の場合について図示し，すべてのプロセスの実行が終了するまでの時間を求めよ。ただし，プロセス実行のために利用できる全メモリ容量は 500kB（OS の使用領域を除く 100kbyte～600kB の領域）とし，P1 から順番に処理を行うものとする。

プロセス	大きさ	実行時間
P1	100kB	10 ミリ秒
P2	250kB	20 ミリ秒
P3	200kB	20 ミリ秒
P4	150kB	25 ミリ秒
P5	250kB	10 ミリ秒

(a) 可変区画割付でコンパクションを行わない場合
(b) 可変区画割付で全体の実行時間が最小になるようにコンパクションを行う場合

(a) の場合



(b) の場合