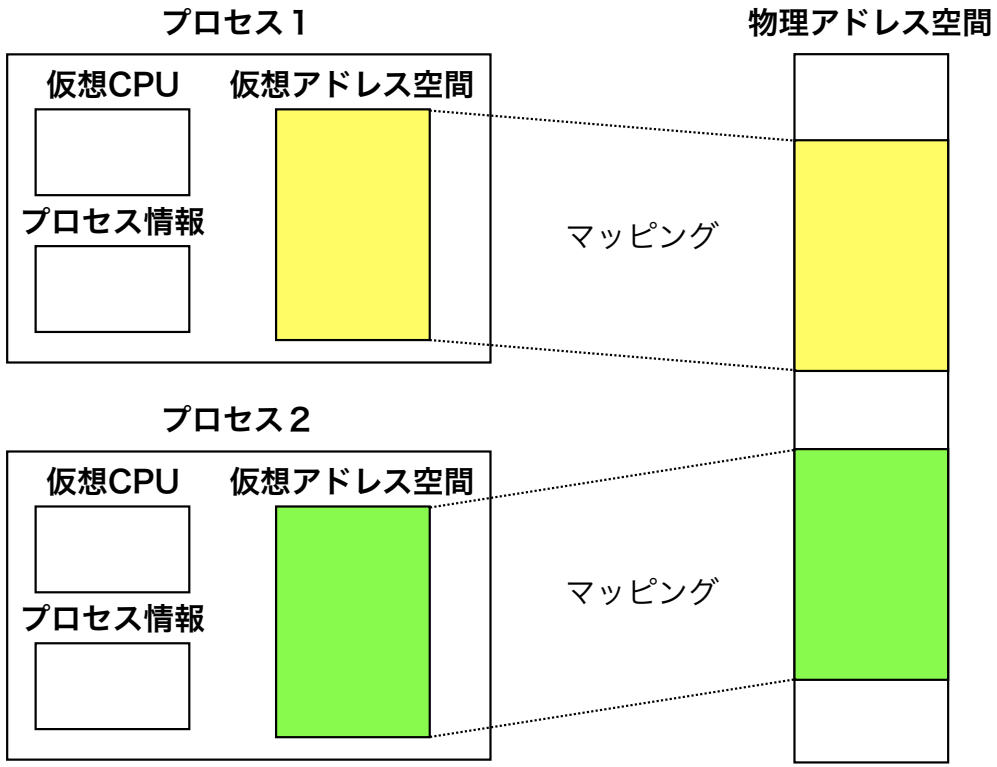
オペレーティングシステム
第１１章 セグメンテーション

https://github.com/tctsigemura/OSTextBook

リロケーションレジスタ方式（復習）

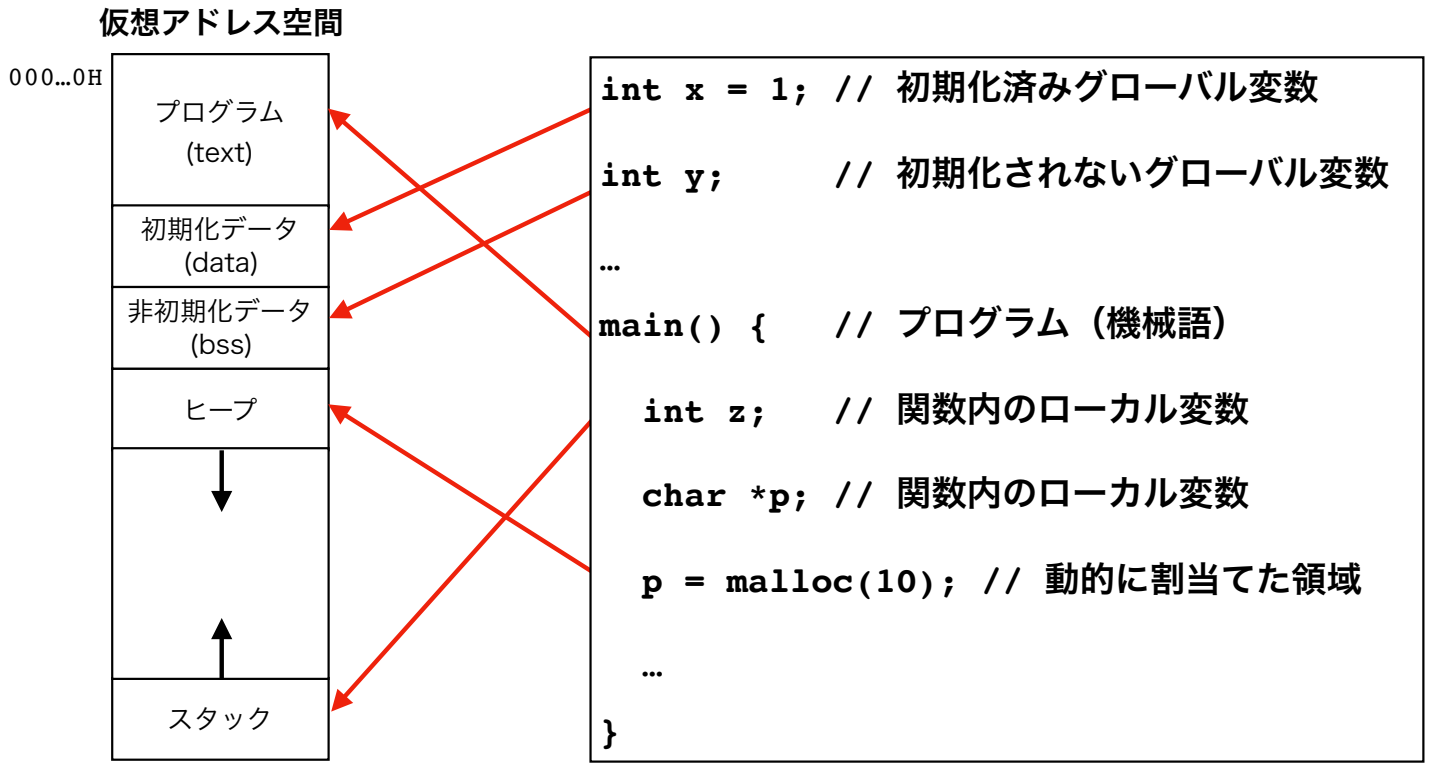
プロセスの仮想アドレス空間を物理アドレス空間にマッピングする．



- プロセス毎に独立した仮想アドレスを持てる．
- 仮想アドレス空間はいつも 0 番地から開始できる．
- 動的なメモリコンパクションができる．

リロケーションレジスタ方式の問題点（1/2）

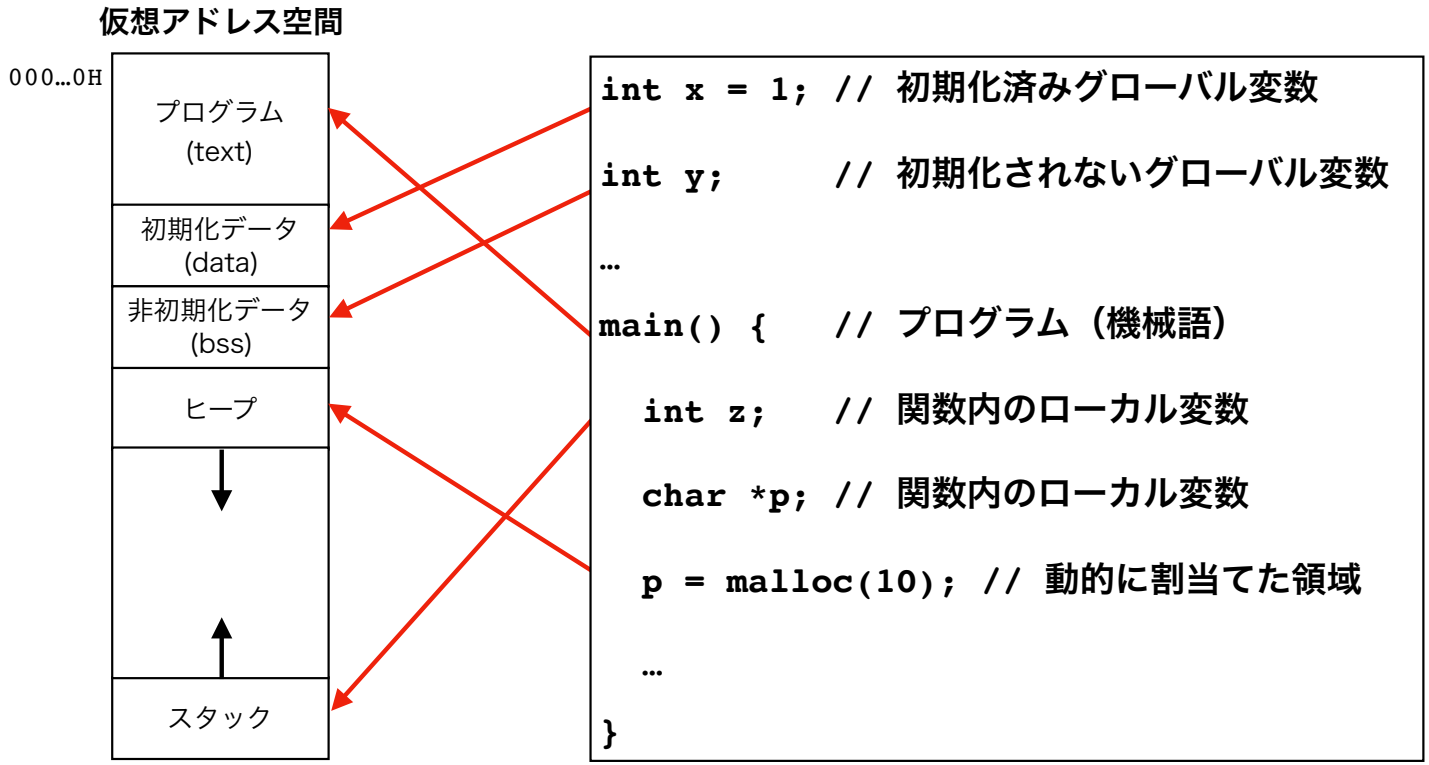
プロセスの仮想アドレス空間は次のような領域から構成された.



- 必要なメモリの見積もりが難しい.
  実行時にヒープ領域やスタック領域が不足する可能性がある.
  実行前に必要なメモリサイズを見積もる必要があり使い勝手が悪い.

リロケーションレジスタ方式の問題点（2/2）

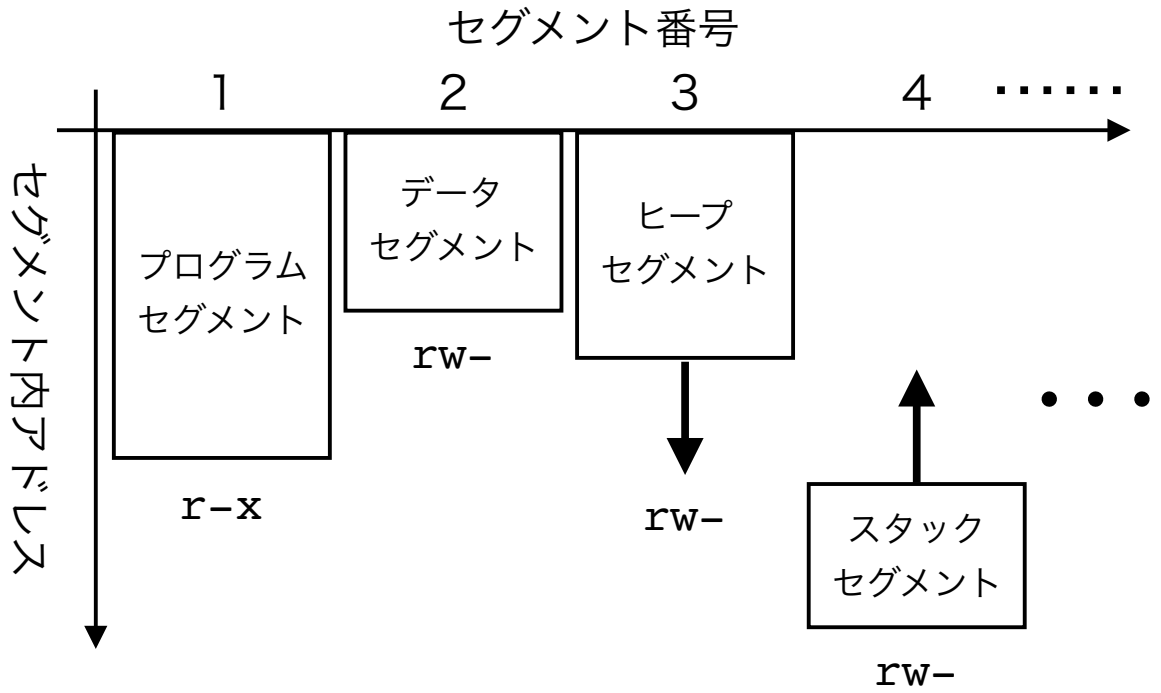
プロセスの仮想アドレス空間は次のような領域から構成された．



- 領域の性質応じたメモリ保護ができない．
  - プログラム領域は読み出しと実行だけ許可する．
  - データ，ヒープ，スタック領域に読み出しと書き込みだけ許可する．

セグメント

プロセスに複数のアドレス空間を持たせる.



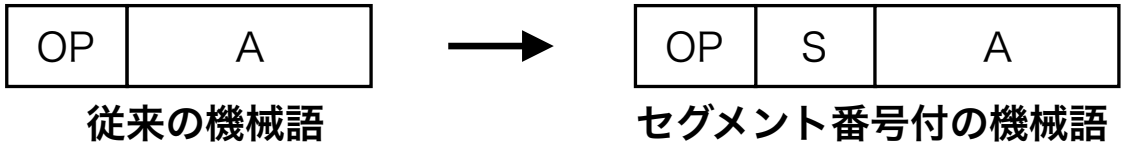
- 複数持つことができるアドレス空間＝**セグメント**
- 前の例で「プログラム」,「データ」等をセグメントにする.
- セグメント毎にサイズや保護モード（rwx）を決める.
- セグメント番号とセグメント内アドレスの二次元空間になった.

仮想アドレス空間内の配置問題が解決！！

セグメント番号

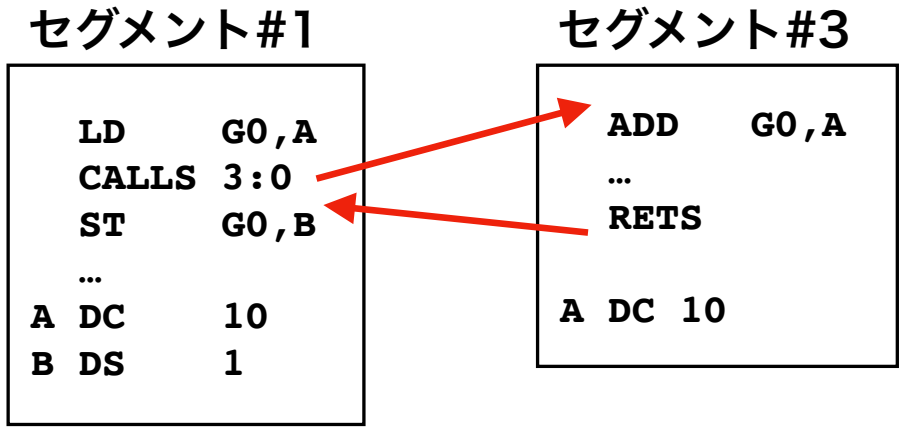
セグメント番号はどこから供給するのか？

- 機械語命令にセグメント番号を持たせる．



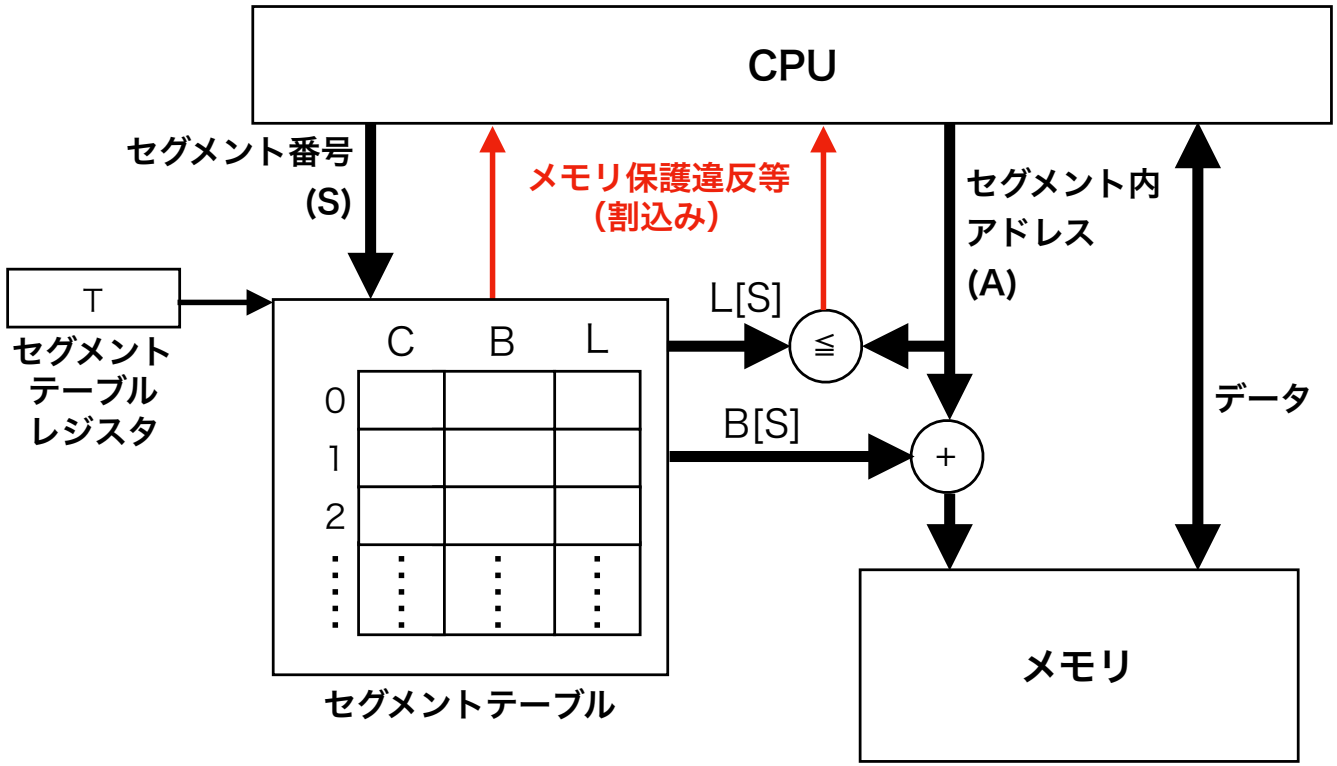
プログラムが大きくなる．

- カレントセグメント



CALLS，RETS 命令でセグメントが自動的に切り換わる．
IA-32 ではカレントセグメントが複数ある．
(CS:プログラム，DS:データ，SS:スタックセグメント等)

セグメンテーション機構（1/3）



- CPU はセグメント番号（S）とセグメント内アドレス（A）を出力
- セグメントテーブルを使用して物理アドレスに変換する.
- 本当は，セグメントテーブルはメモリ上に置く.
- セグメントテーブルレジスタはセグメントテーブルのアドレス

セグメンテーション機構（2/3）

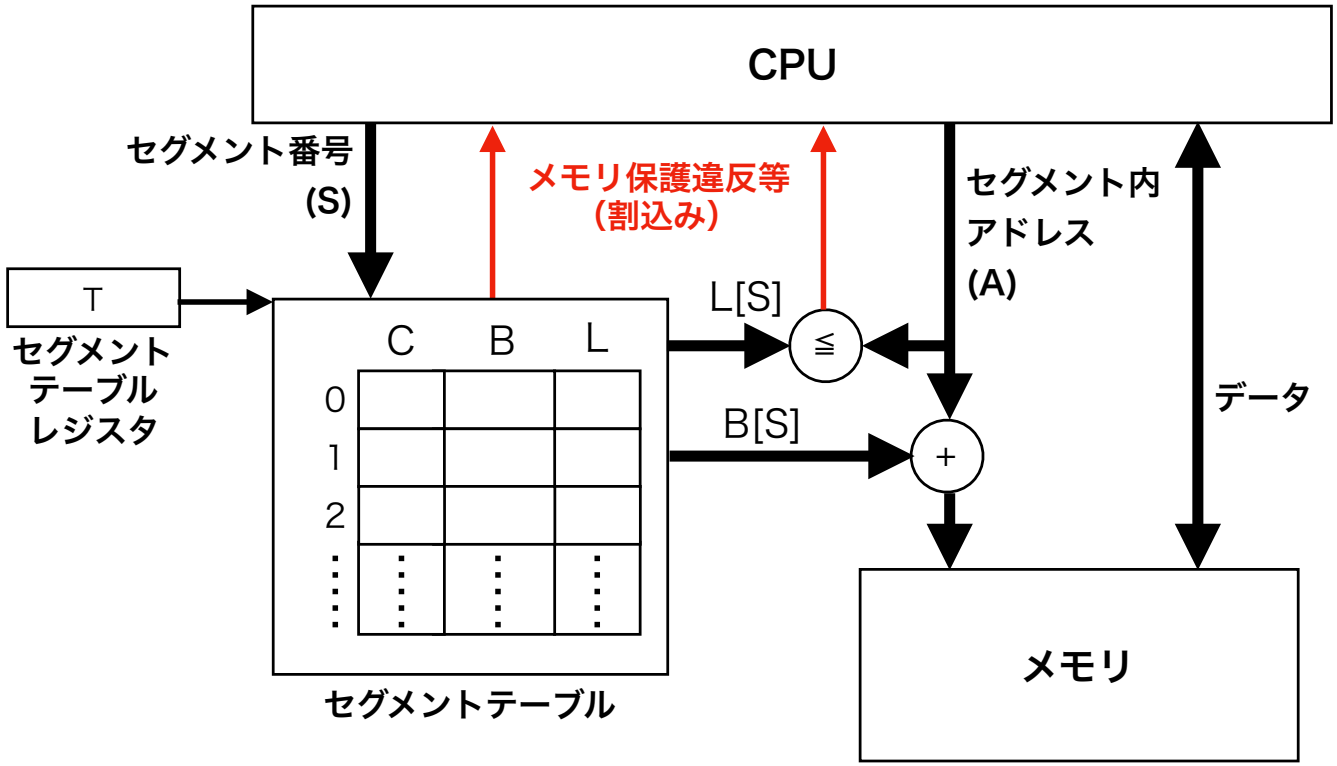
セグメントテーブル

- セグメント番号をインデクスにしてSでエントリを選択する.
- B（Base）フィールドはセグメントのベースアドレス
- L（Limit）フィールドはセグメントのサイズ
- BとLはリロケーションレジスタと同じ.
- C（制御）フィールドは以下のビットを含む.

名称	ビット数	意　味
V（Valid）	1	メモリにロードされている.
D（Dirty）	1	ロード後に変更された.
RWX（Read/Write/eXecute）	3	許されるアクセス方法.

V=0 なら**セグメント不在割込み**を発生
D=0 ならスワップアウトで書き戻しが不要
RWX はそのセグメントに許される操作（**メモリ保護違反割込み**）

セグメンテーション機構（3/3）



物理アドレスへの変換

1. セグメントテーブルのエントリを読み出す.
2. C フィールド（V，RWX）をチェックする.
3. セグメント内アドレスをチェックする.
4. $B[S] + A$ が物理アドレスになる.

セグメントテーブルエントリのキャッシング

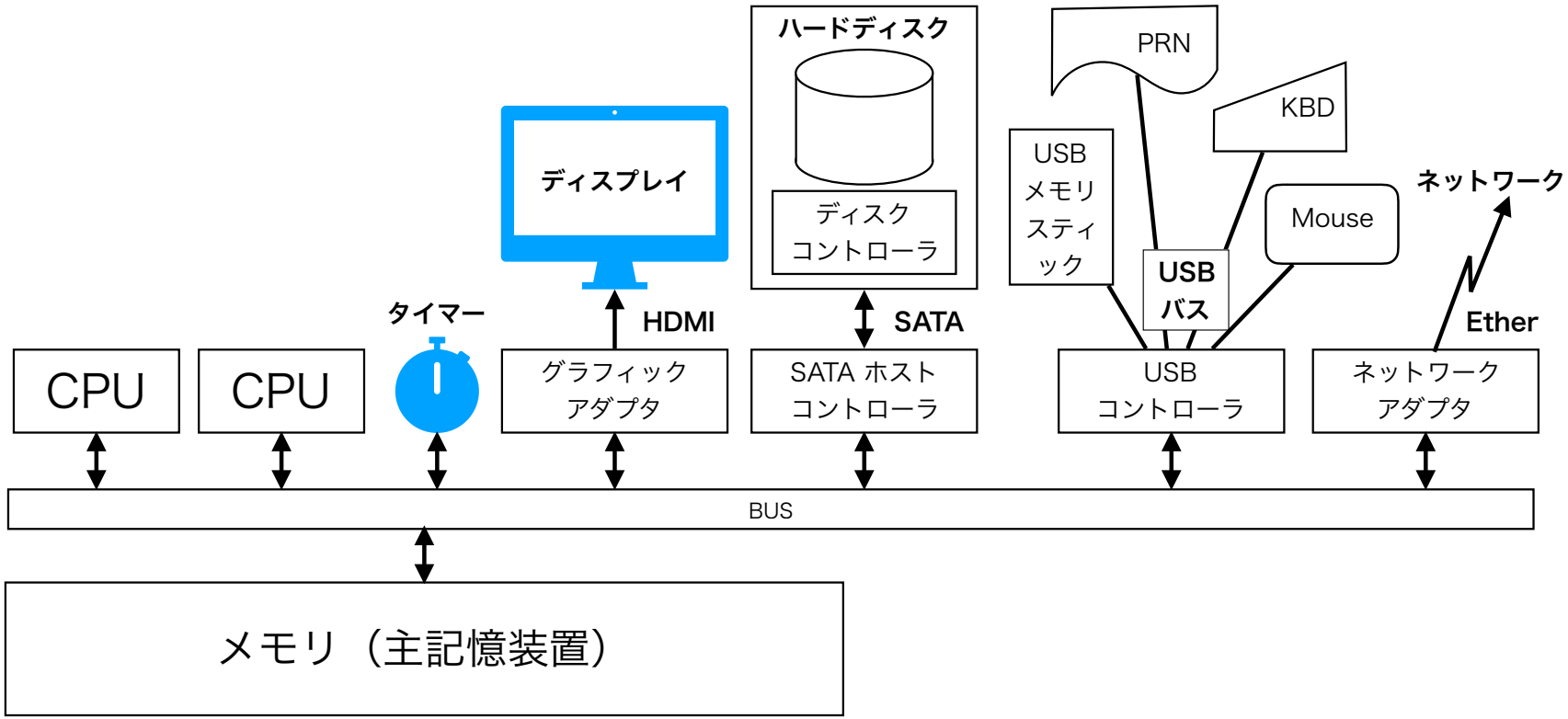
物理アドレスへの変換

- メモリアクセスの度にセグメントテーブルを参照？
- メモリアクセス回数が二倍になる．
- メモリはいつも混み合っている（次のページ）．
- 一度参照したセグメントテーブルエントリは CPU または MMU にキャッシュするべきだ．
- IA-32 ではカレントセグメントレジスタに裏レジスタがある．

セグメントレジスタ		裏レジスタ		
CS	セレクタ	ベース	リミット	属性
DS	セレクタ	ベース	リミット	属性
SS	セレクタ	ベース	リミット	属性
ES	セレクタ	ベース	リミット	属性
FS	セレクタ	ベース	リミット	属性
GS	セレクタ	ベース	リミット	属性

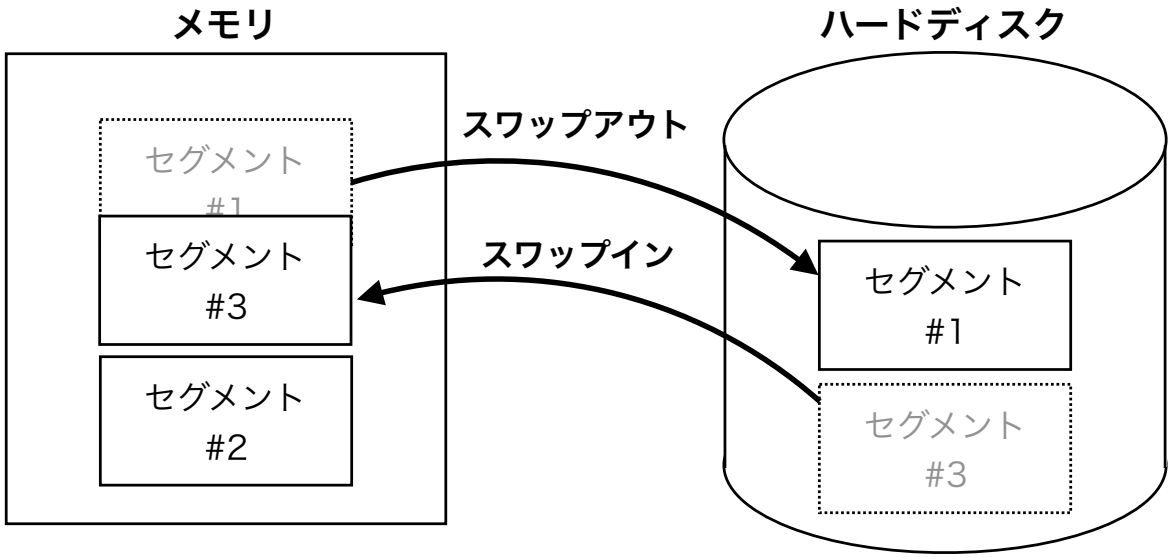
裏レジスタは自動的にロード，使用される．

メモリはいつも混み合っている（参考）



- 命令もデータも全てメモリにある.
- CPU も I/O 装置もメモリにアクセスする.
- フォン・ノイマン・ボトルネック (Von Neumann bottleneck)

フラグメンテーション機構による仮想記憶



全体がメモリに収まらない大きなプログラムも実行できる.

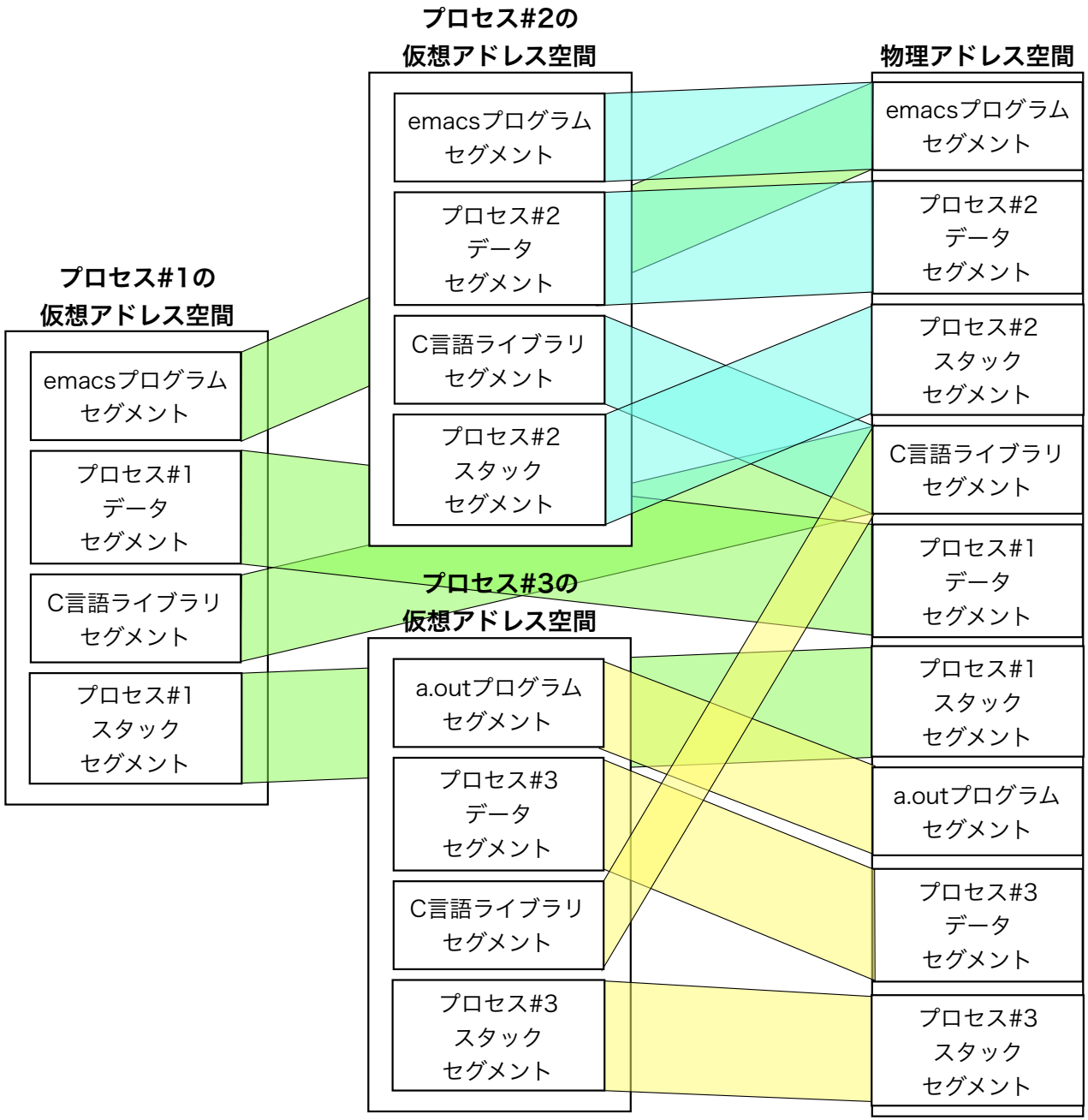
- 例えばセグメント 3 のサブルーチンを実行したい（CALL 3:0）
- セグメント 3 はセグメント不在なので OS に切り換わる.
- メモリに入り切らないのでセグメント 1 を**スワップアウト**
- メモリの空き領域にセグメント 3 を**スワップイン**

知らない間に OS が自動的にスワッピングを行う.（仮想記憶！！）

セグメントの共用

プロセス間でセグメントを共用しメモリを節約する．

- プログラムや定数等は書き変わらない．(共有可)
  - プログラム本体
  - サブルーチン（ライブラリ）
  - 定数表
- データ，ヒープ，スタック等は書き換わる．(共有不可)
  プロセス毎に別のコピーを持つ必要がある．


プロセス#1の
仮想アドレス空間
emacsプログラム
セグメント
プロセス#1
データ
セグメント
C言語ライブラリ
セグメント
プロセス#1
スタック
セグメント
プロセス#2の
仮想アドレス空間
emacsプログラム
セグメント
プロセス#2
データ
セグメント
C言語ライブラリ
セグメント
プロセス#2
スタック
セグメント
プロセス#3の
仮想アドレス空間
a.outプログラム
セグメント
プロセス#3
データ
セグメント
C言語ライブラリ
セグメント
プロセス#3
スタック
セグメント
物理アドレス空間
emacsプログラム
セグメント
プロセス#2
データ
セグメント
プロセス#2
スタック
セグメント
C言語ライブラリ
セグメント
プロセス#1
データ
セグメント
プロセス#1
スタック
セグメント
a.outプログラム
セグメント
プロセス#3
データ
セグメント
プロセス#3
スタック
セグメント

セグメンテーションの利点・欠点（1/2）

利点

- セグメントには，例えば「C 言語ライブラリセグメント」のような，論理的な意味を持たせることができる．
- セグメントの論理的な意味を反映したメモリ保護が可能である．
- プログラムやデータの共用が容易である．
- セグメントの長さは自由に決められるので内部フラグメントが発生しない．
- セグメントの長さは動的に変化させることも可能である．
- セグメント単位のスワッピングを用いて仮想記憶を実現できる．

セグメンテーションの利点・欠点（2/2）

欠点

- 物理アドレス空間に外部フラグメントが生じる.
- 外部フラグメントの解消にはメモリコンパクションが必要である.
- 物理メモリ上に連続した領域が必要である.
- 物理メモリより大きいセグメントを作ることができない.

これらの欠点を克服するために，ページングを組み合わせたシステムがある.（IA-32，MULTICS）

問題

- セグメンテーション機構の図で，セグメントテーブルに適当な値を設定し幾つかの二次元アドレスが変換させる物理アドレスを確かめなさい.
- あるセグメントサイズが大きくなる場合の，セグメントテーブルの変更項目等を指摘しなさい.
- メモリコンパクションの手順を説明しなさい.
- スタックセグメントを意識した**前向きに伸びるセグメント**も利用可能なセグメンテーション機構を設計しなさい.
  1. セグメントテーブルに必要な変更は？
  2. 機構に必要な変更は？
  3. 他に必要な変更は?