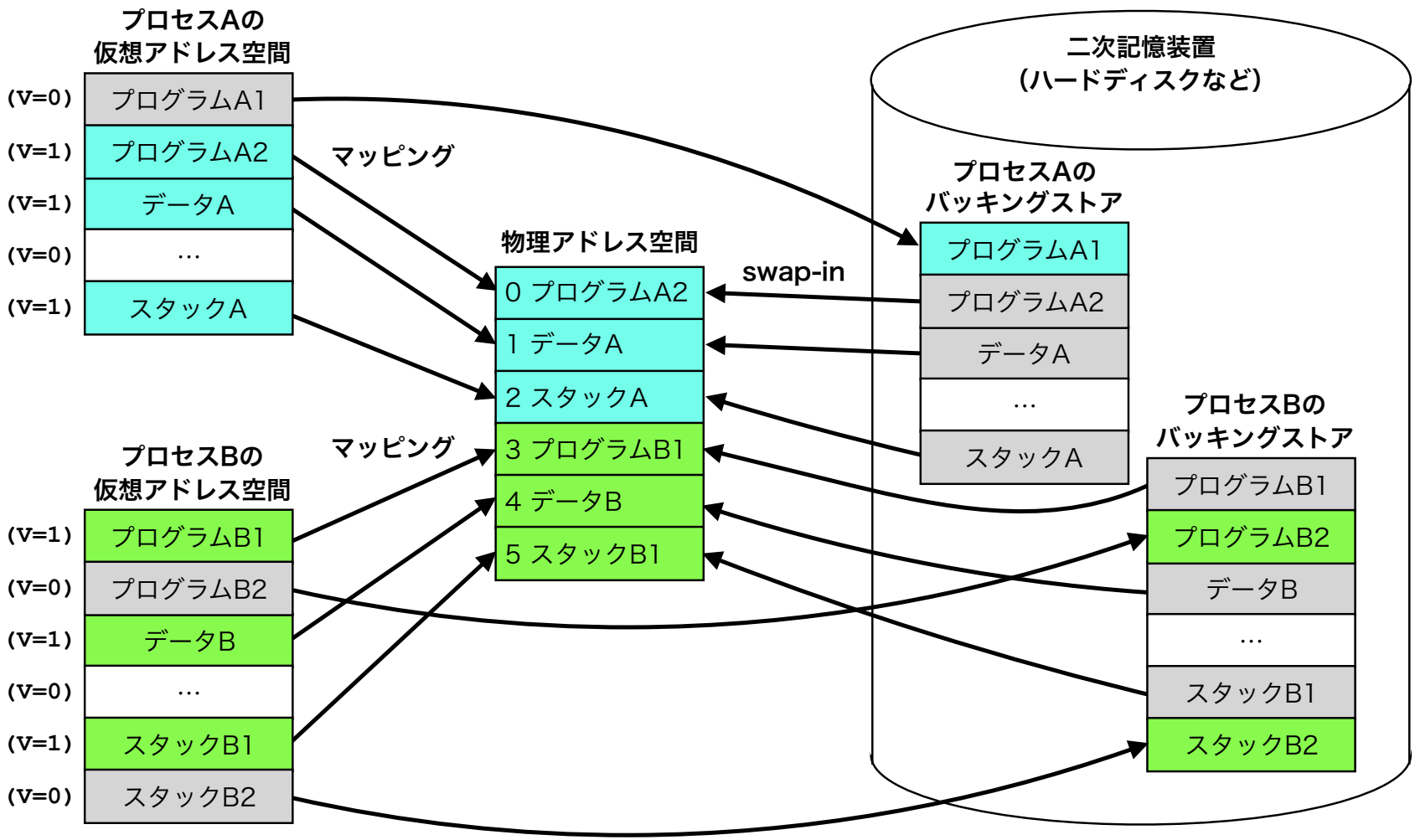
オペレーティングシステム
第１３章 仮想記憶

基本概念

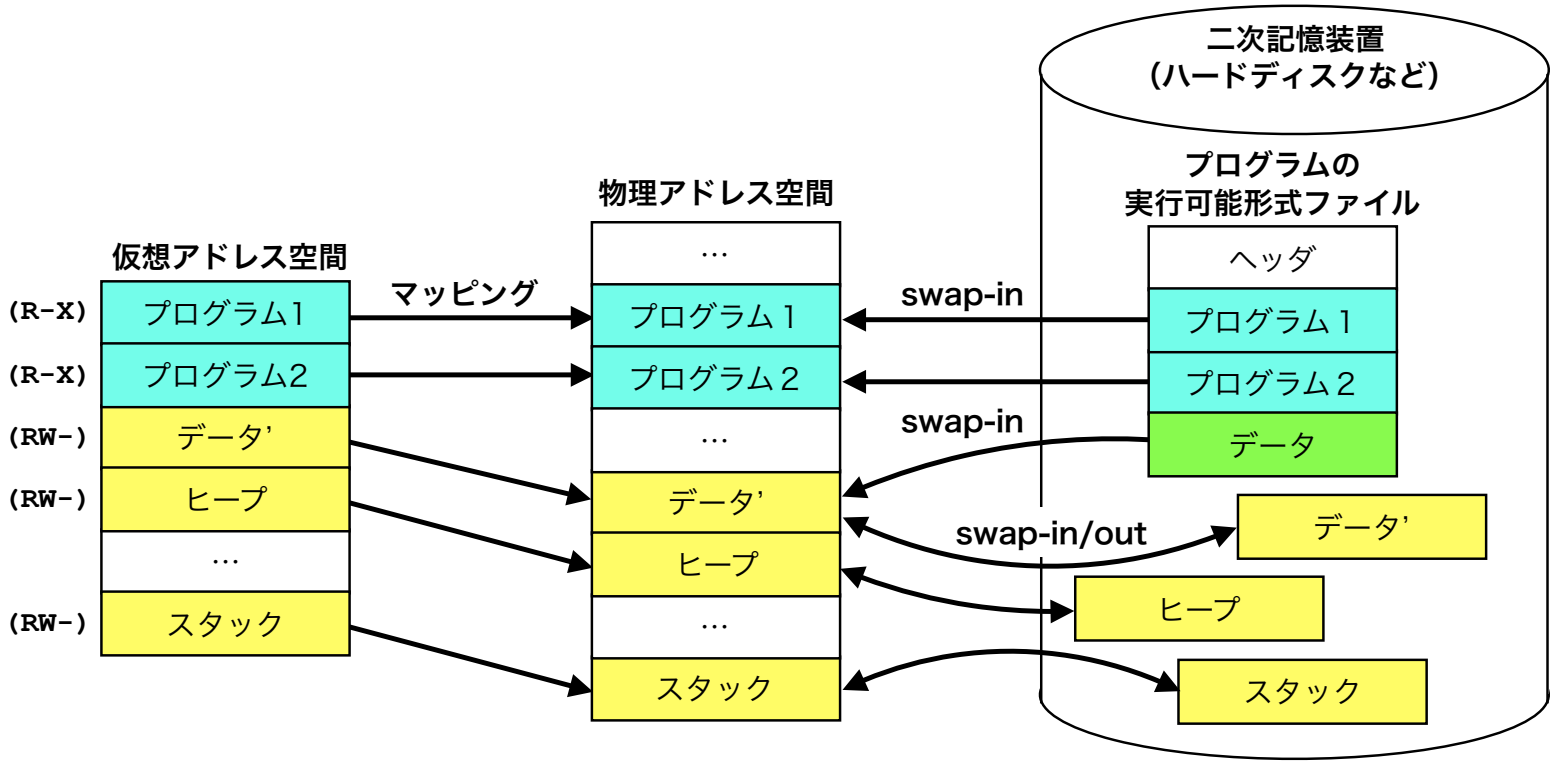
ページングをベースに仮想記憶を実現する.

- システムの使用メモリ合計が物理メモリより大きい. → 実行可
- 単一のプログラムがメモリより大きい。→ 実行可
- ページテーブルの V=0 を上手く使用する.
- V=0 のページにアクセスするとページ不在割込み → OS へ
- プロセステーブルの V=0 に二つの場合がある.
  1. 無効な領域 → プロセス終了
  2. **バッキングストア**に退避中 → 復旧して再開
- プロセス生成時にバッキングストアにプロセスのイメージを作る.
- Windows，macOS，Linux 等，現代の OS のほとんどが採用している.

仮想記憶の基本

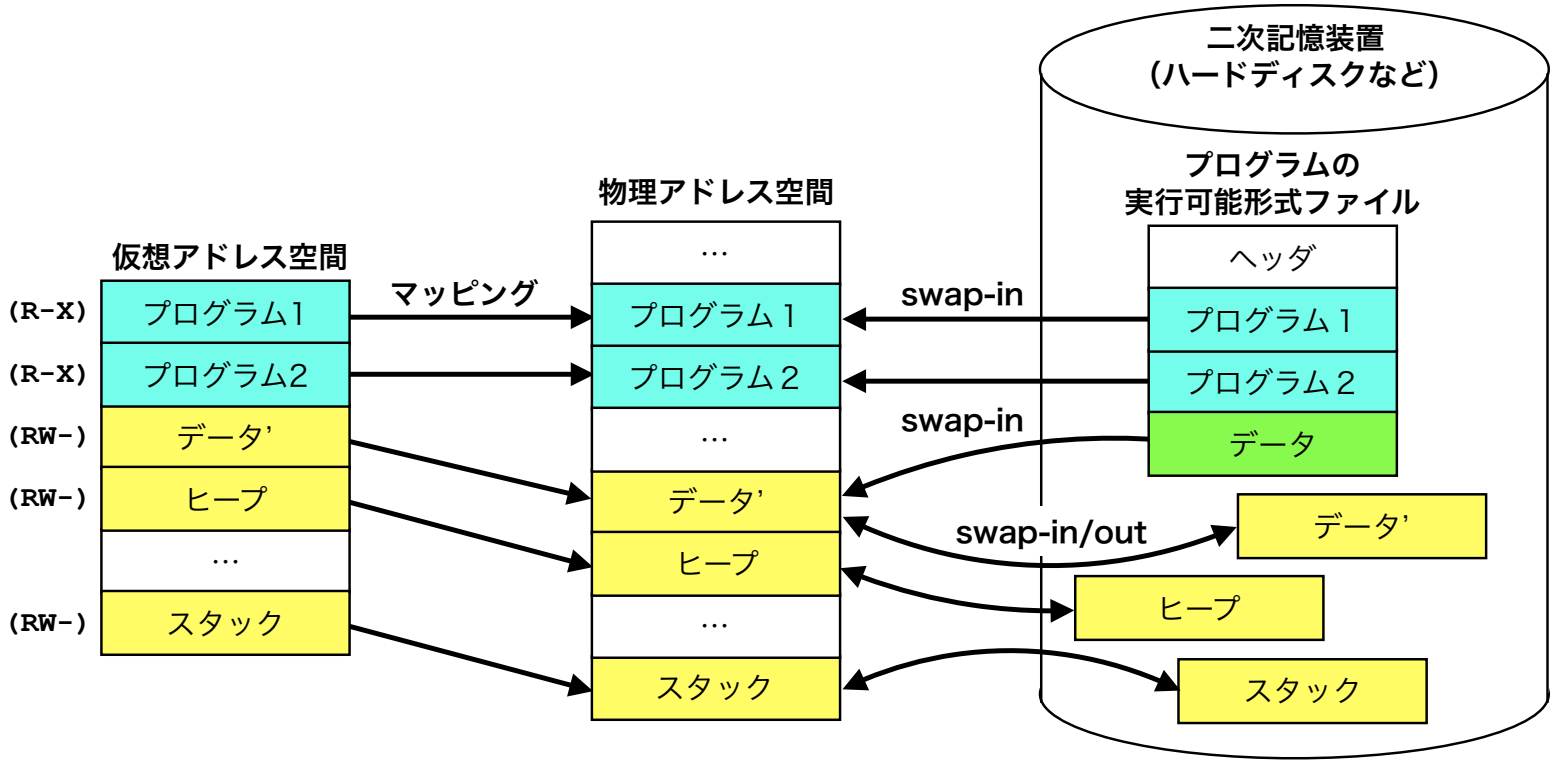

プロセスAの
仮想アドレス空間
(V=0) プログラムA1
(V=1) プログラムA2
(V=1) データA
(V=0) …
(V=1) スタックA
マッピング
物理アドレス空間
0 プログラムA2
1 データA
2 スタックA
3 プログラムB1
4 データB
5 スタックB1
swap-in
二次記憶装置
(ハードディスクなど)
プロセスAの
バッキングストア
プログラムA1
プログラムA2
データA
…
スタックA
プロセスBの
バッキングストア
プログラムB1
プログラムB2
データB
…
スタックB1
スタックB2
マッピング
プロセスBの
仮想アドレス空間
(V=1) プログラムB1
(V=0) プログラムB2
(V=1) データB
(V=0) …
(V=1) スタックB1
(V=0) スタックB2

デマンドページング（Demand Paging）



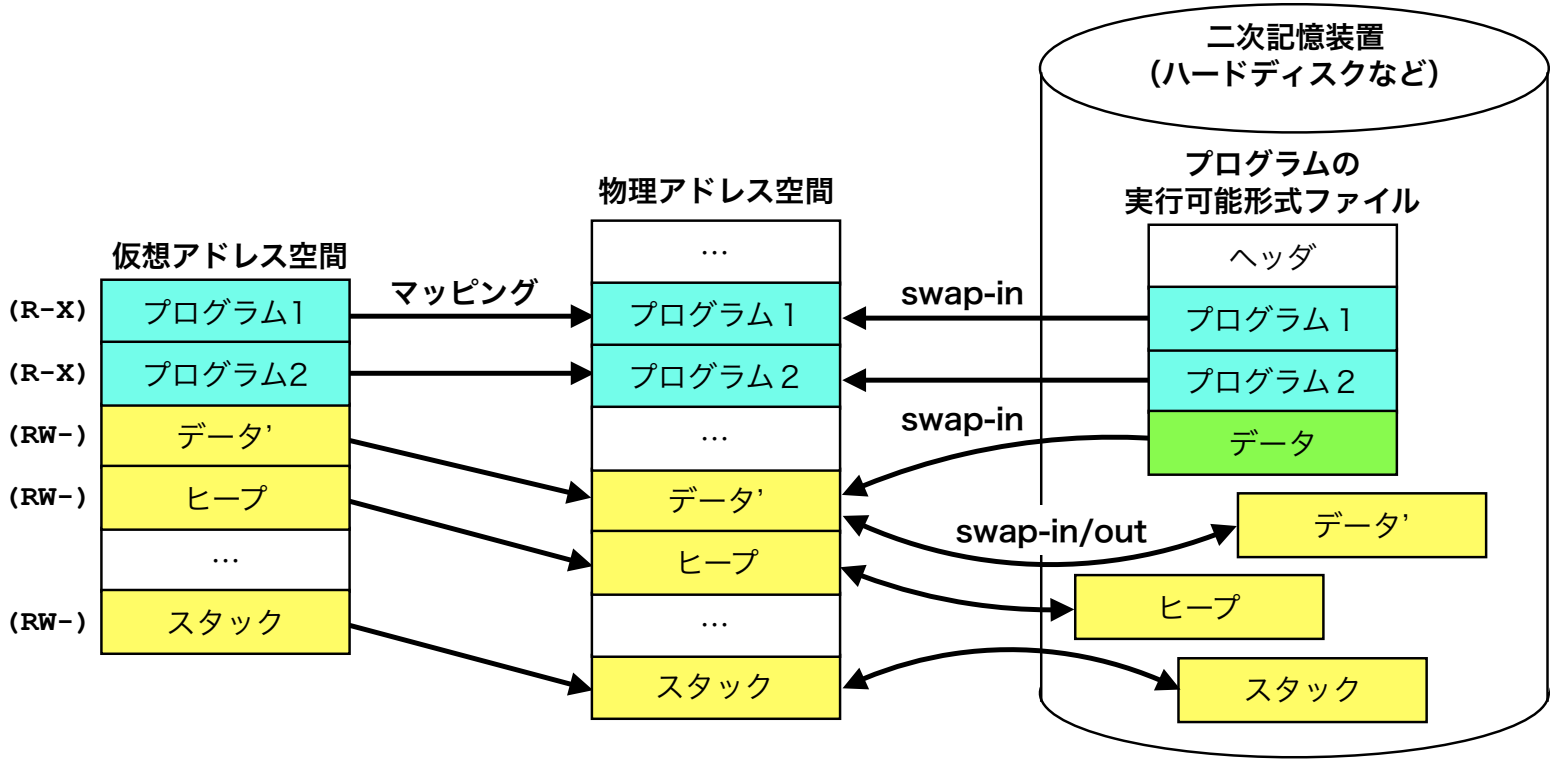
- ページを swap-in するための方式の一つ.
- 全てのページが不在の状態からスタートする.
- ページ不在を起こしたページを swap-in する.
  (使用しないページを読み込むような無駄が無い)

プログラムファイルの直接 swap-in による実行



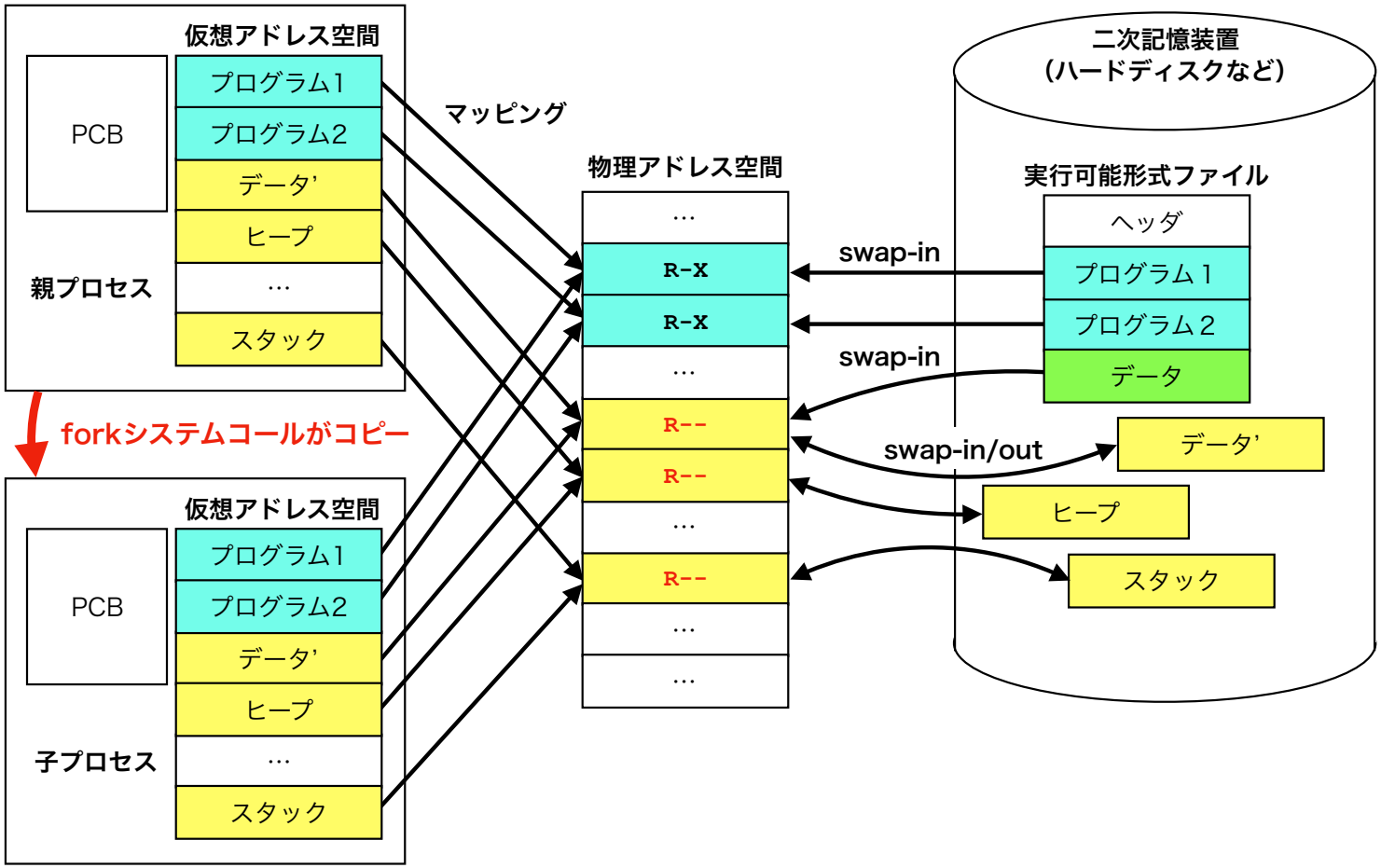
- デマンドページング用の実行可能形式ファイルを用いる．
  (このファイルはページサイズを意識した構造になっている)
- プログラムはファイルから swap-in する（R-X に設定)．
- 初期化データはファイルから swap-in する（RW-に設定)．
- 非初期化データ，ヒープ，スタックはゼロにする（RW-に設定)．

プログラムの swap-out



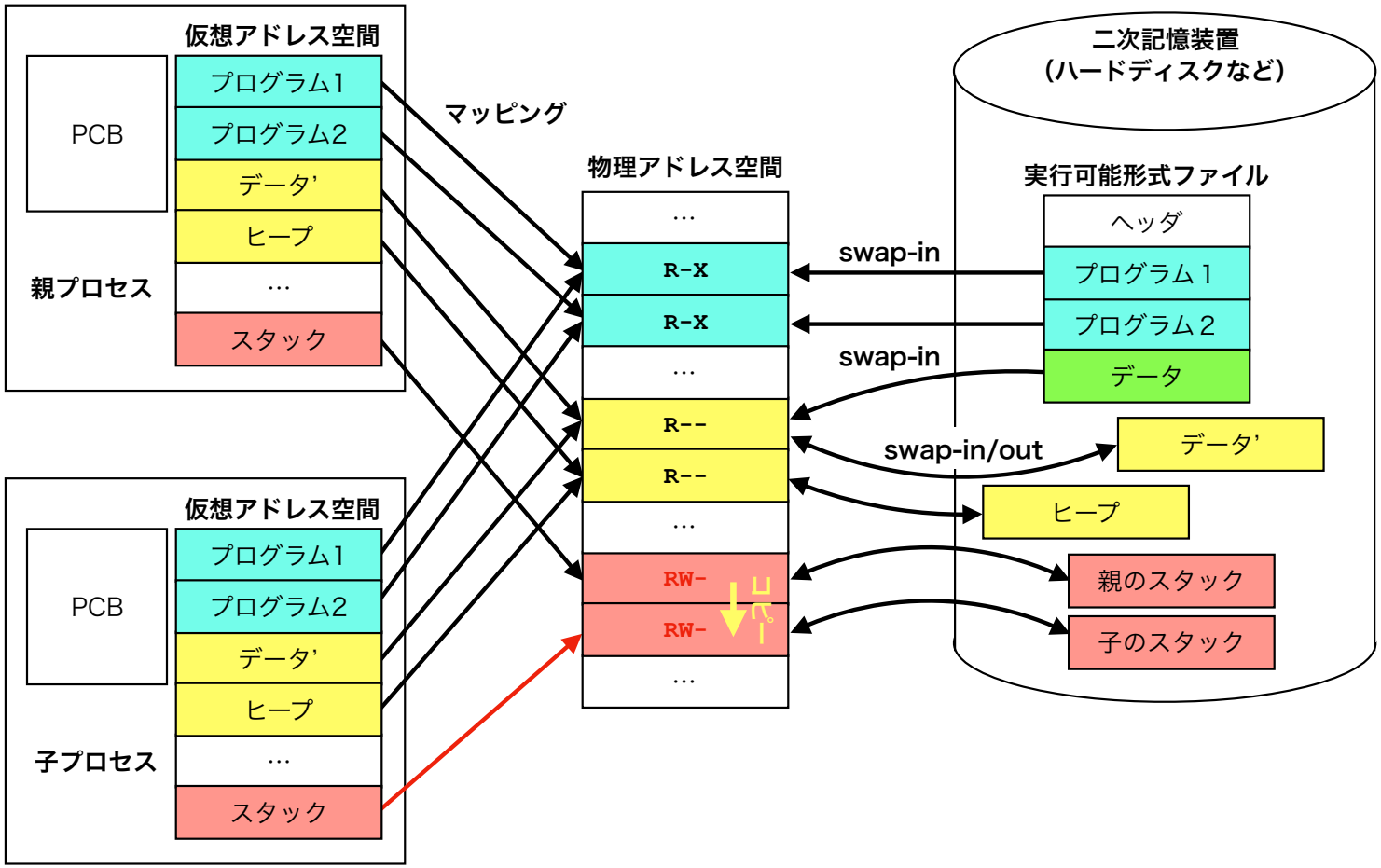
- フレームが枯渇したら使用頻度の低いフレームを解放し再利用する.
- プログラム（R-X）は変化しないので swap-out しない.
- 初期化データ（RW-）はバッキングストアに swap-out する.
- 非初期化データ，ヒープ，スタックも swap-out する.

Copy on Write（1）



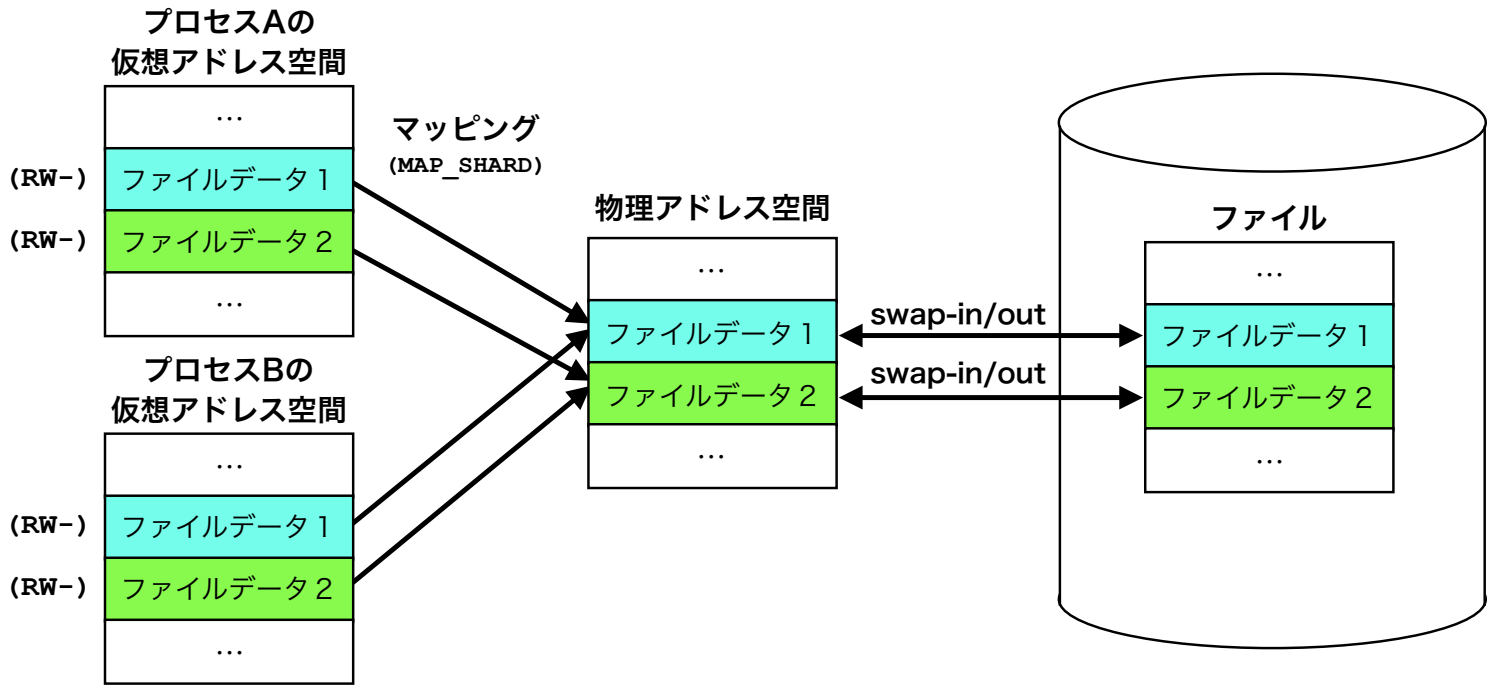
- fork-exec ではアドレス空間のコピーに無駄が多い．→ vfork
- vfork は使いにくい．使いやすい fork を改良する．
- fork の後，書き込み可能ページを一時的に R--に設定しておく．

Copy on Write（2）



- 例えばスタックに**書き込み**があるとメモリ保護割込みが発生する.
- この時点で OS が新しいフレームを割当て，内容を**コピー**する.
- ページを RW-に変更しプロセスを再開する.

メモリマップドファイル（1）



- 仮想記憶機構を用いたファイルへのアクセス手段である.
- プロセスはメモリ上の配列のようにファイルにアクセスできる.
- ファイルアクセスで，一々システムコールを使用しない.
  (軽いファイルアクセス手段)
- 同じファイルを複数プロセスがマッピング → 共有メモリになる.

メモリマップドファイル（2）

UNIX のメモリマップドファイルの例（mmap システムコール）

```
void * mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);
```

戻り値： マップされた領域の先頭アドレスが返される.

addr： マップしたい仮想アドレス空間の先頭アドレスを渡す.

len： マップする領域の大きさを渡す.

prot： 保護モード（protection：RWX）を表す値を渡す.

flags： 共用する（MAP_SHARED）／しない（MAP_PRIVATE）等

fd： オープン済みファイルのファイルディスクリプタを渡す.

offset： ファイル中のマッピング位置.

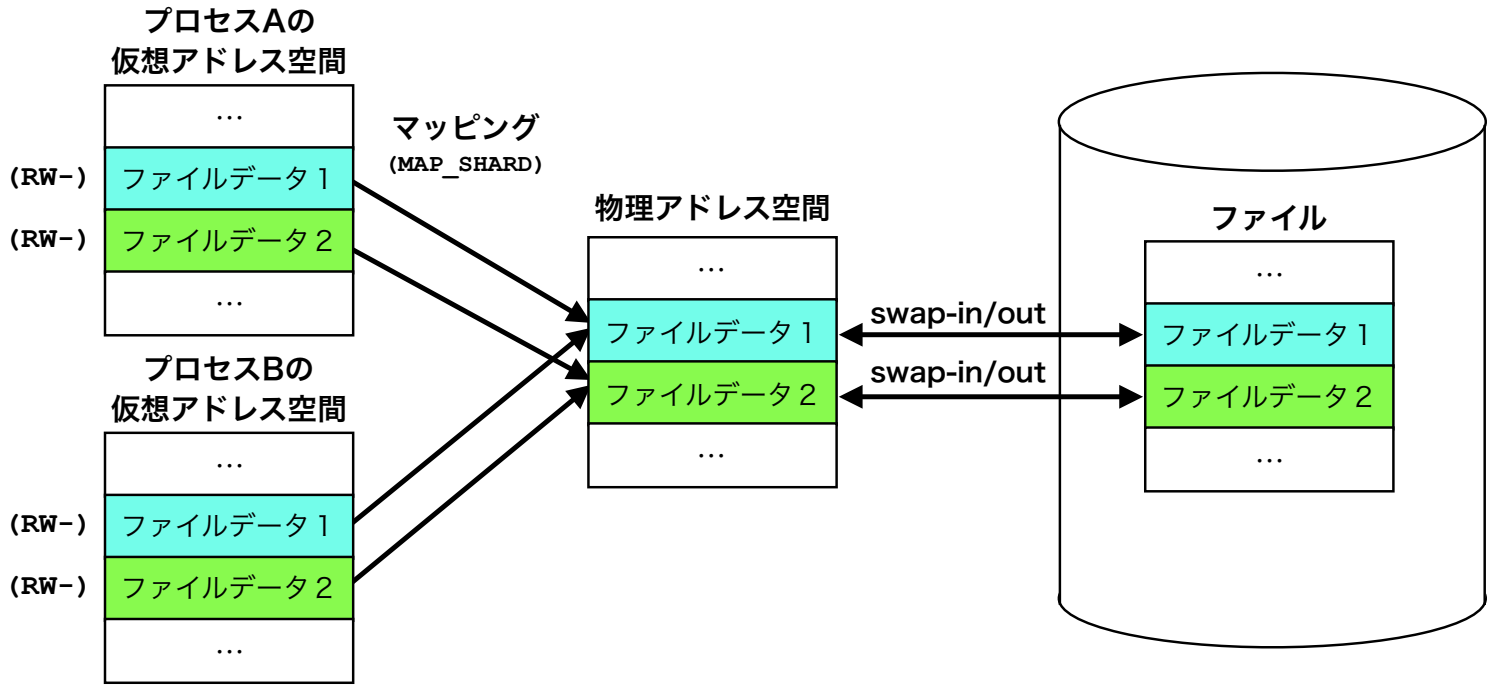
アドレスや長さはページサイズの整数倍にする.

メモリマップドファイル（3）

```
#include <stdio.h>                      // perror のために必要
#include <fcntl.h>                      // open のために必要
#include <unistd.h>                     // close のために必要
#include <sys/mman.h>                   // mmap のために必要
int main() {
  int fd;
  char *p, *fname="a.txt";
  fd = open(fname, O_RDWR);             // 予め作成してある 4KiB のファイルを開く
  if (fd<0) {
    perror(fname);
    return 1;
  }
  p = mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);  // マップする
  if (p==MAP_FAILED) {
    perror("mmap");
    return 1;
  }
  close(fd);                            // マップしたらクローズして良い
  for (int i=0; i<4096; i++) {          // ファイルに A〜Z を繰り返し書き込む
    p[i] = 'A' + (i % 26);
  }
  return 0;
}
```

メモリマップドファイル（4）

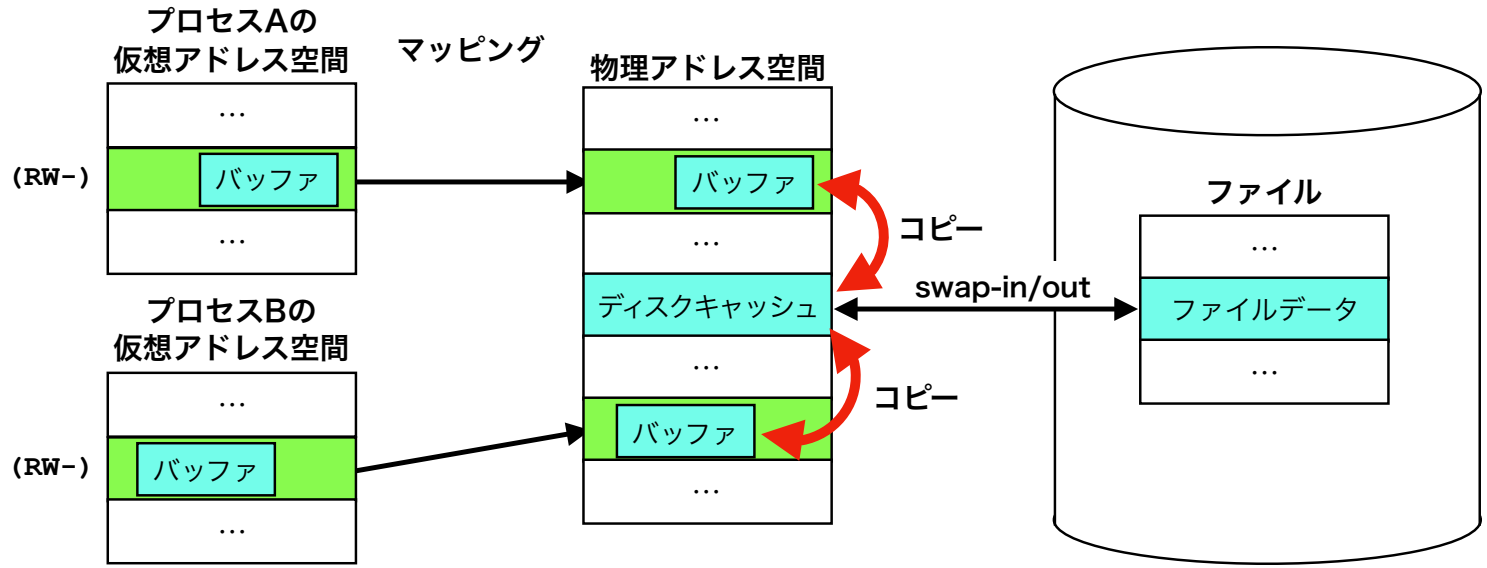
メモリマップドファイルの仕組み



- ファイルの読み込みはデマンドページングの要領で行う.
- ファイルの書き込みは
  - Dirty ページを定期的にファイルに書き戻す.
  - プロセスの終了やマッピングの解消時に書き戻す.

メモリマップドファイル（5）

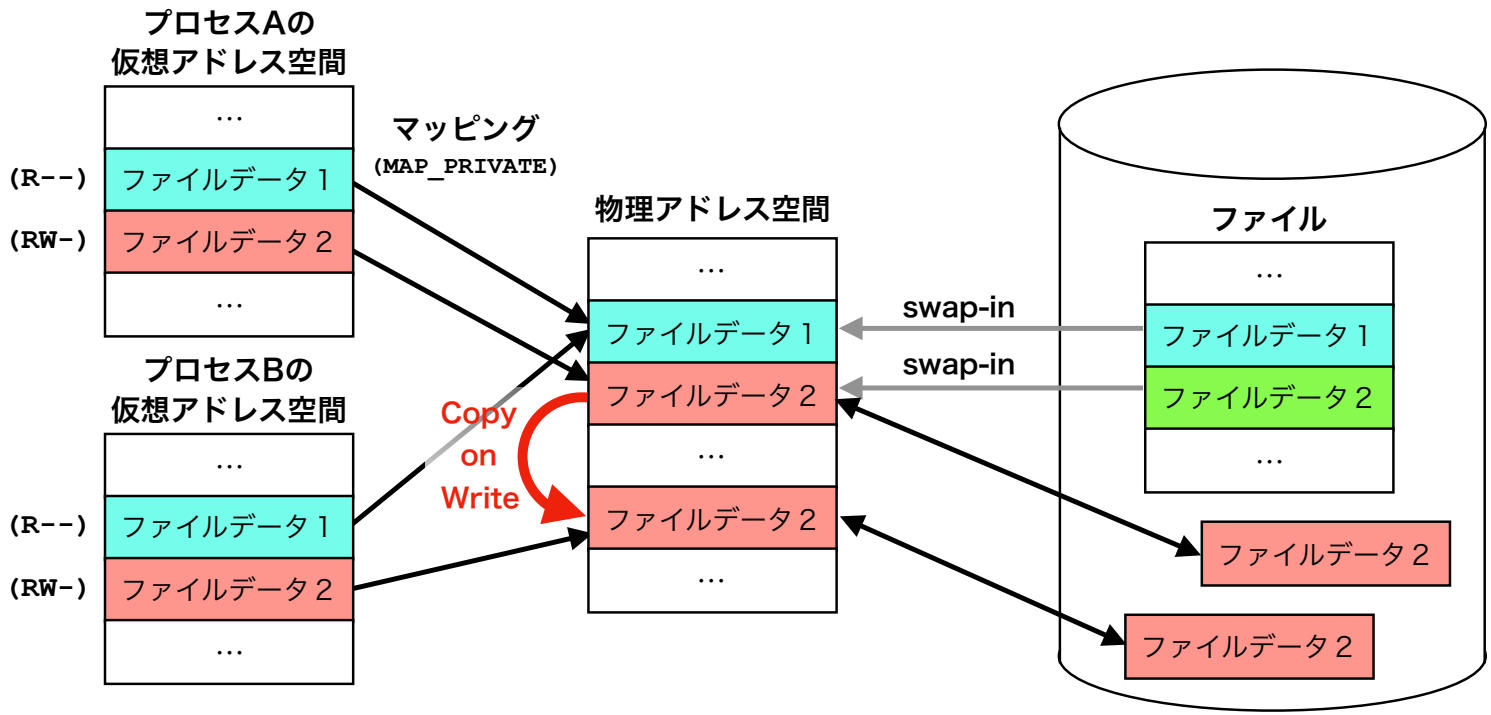
read/write システムコールとの比較



- ファイルを操作する度にシステムコールを発行する.
  (システムコールは重い処理)
- ディスクキャッシュとプログラムのバッファ間でメモリコピー
  (メモリコピーは重い処理)

メモリマップドファイル（6）

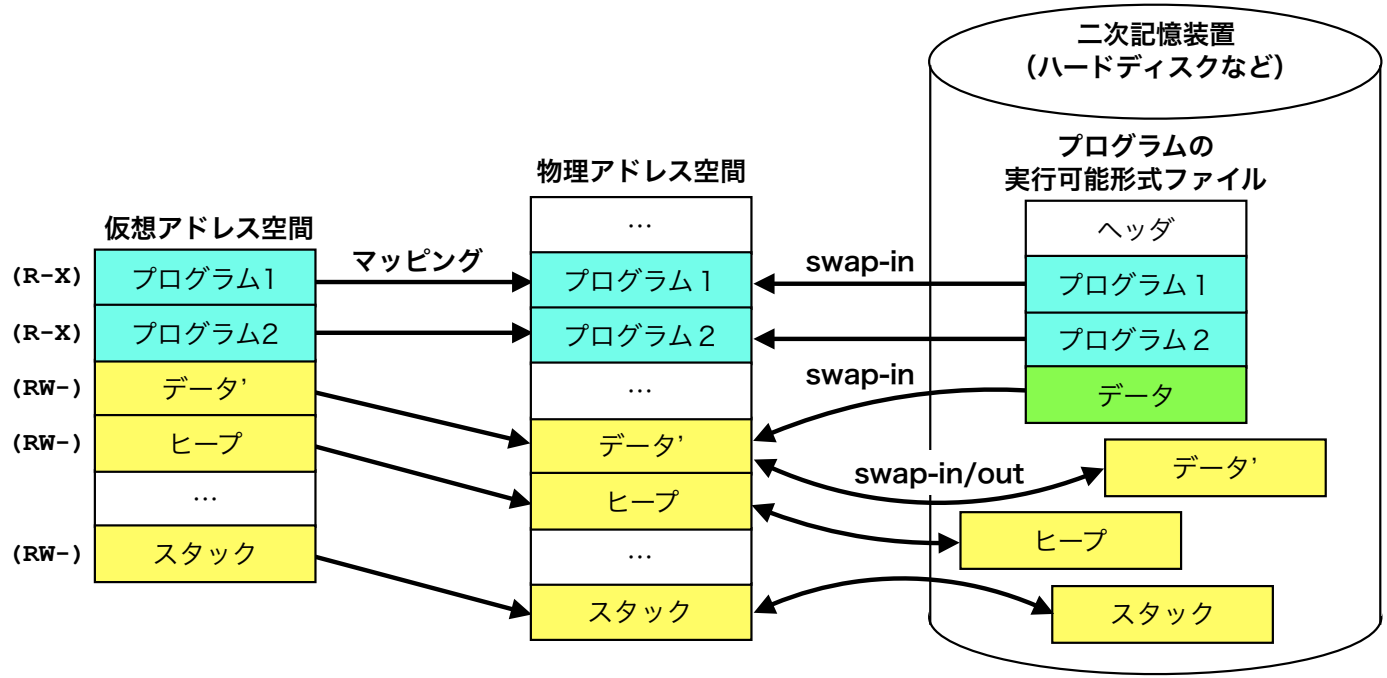
プロセスにローカルなマッピング



- これまでは MAP_SHARD の例だった.
- MAP_PRIVATE の例を紹介する.
- 最初は「ファイルデータ１」のように共有される（R--）.
- 書き換えが発生した時点でコピーを作る（Copy on Write）.
- 「ファイルデータ２」のようにプロセスは別々のコピーを参照する.

メモリマップドファイル（7）

プログラムの実行とメモリマップドファイル



- 実行形式ファイルをメモリにマッピングする．
- プログラムは，R-X，MAP_SHARD でマッピングする．
  （プログラムはプロセス間で共用される）
- 初期化データは，RW-，MAP_PRIVATE でマッピングする．
- 非初期化データ，ヒープ，スタックはファイルにマッピングしない．

練習問題

(1) 次の言葉の意味を説明しなさい.

- 仮想記憶
- デマンドページング
- swap-in，swap-out
- Copy on Write
- メモリマップドファイル