オペレーティングシステム
第１２章 仮想記憶

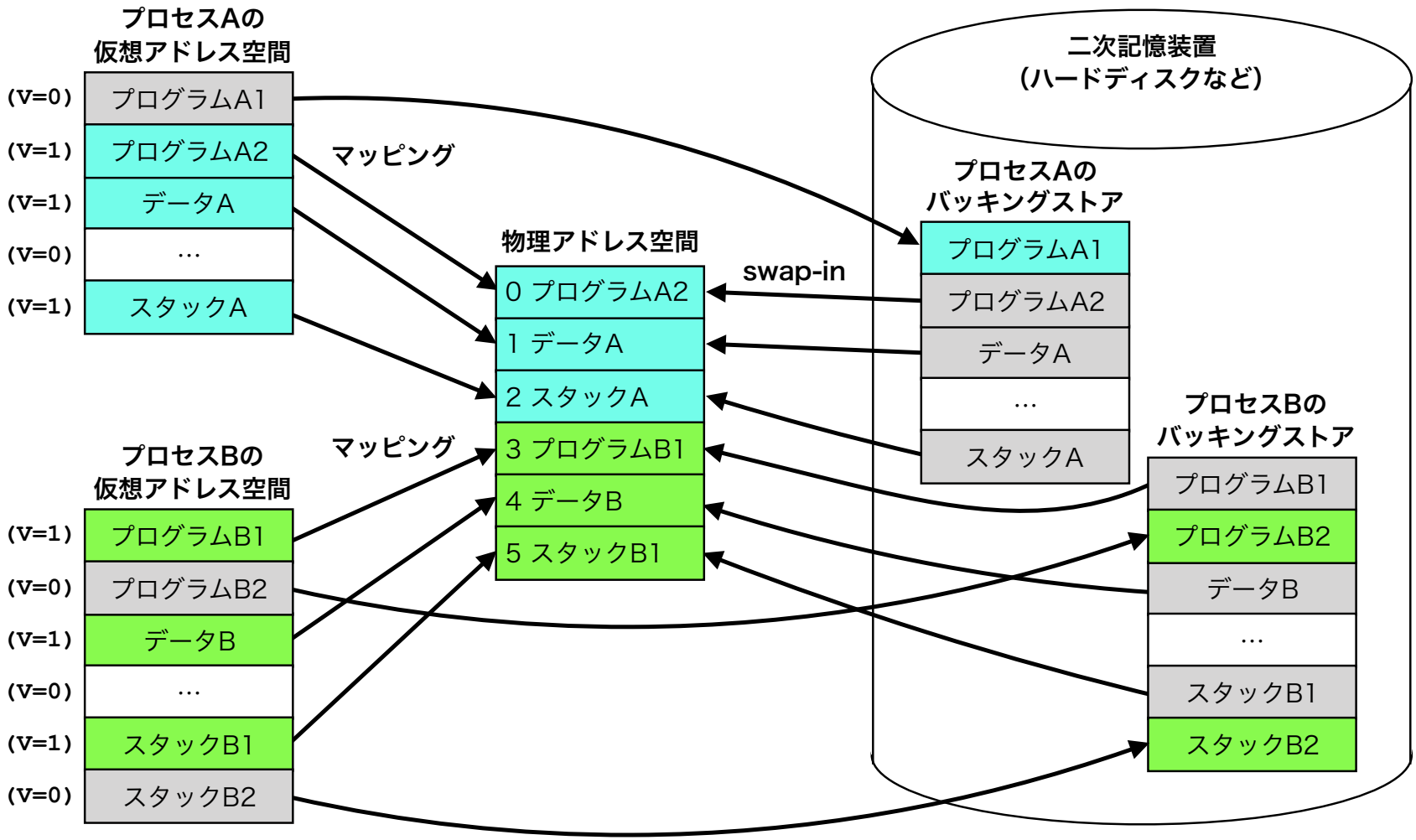
https://github.com/tctsigemura/OSTextBook

基本概念

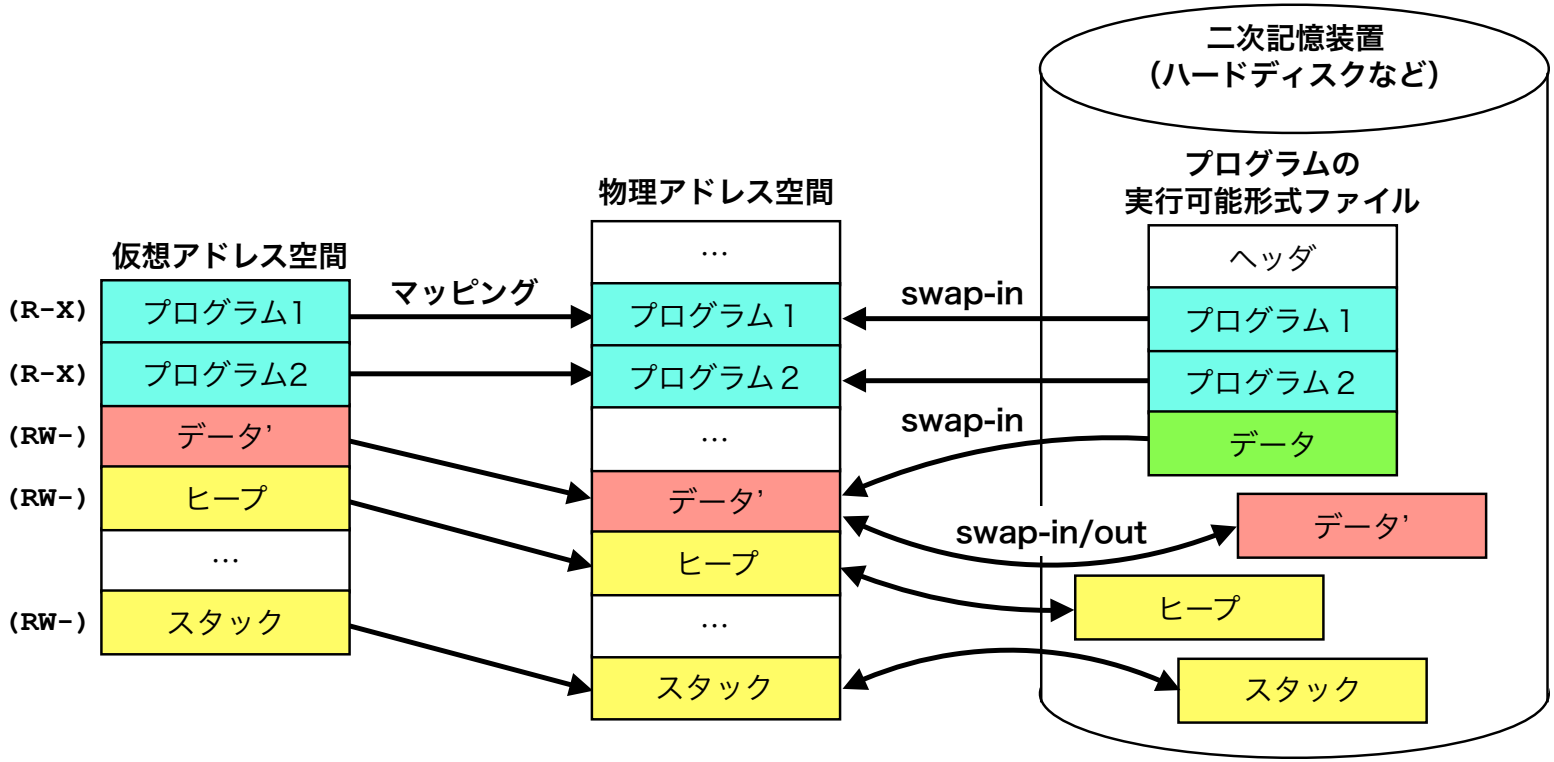
ページングをベースに仮想記憶を実現する.

- 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

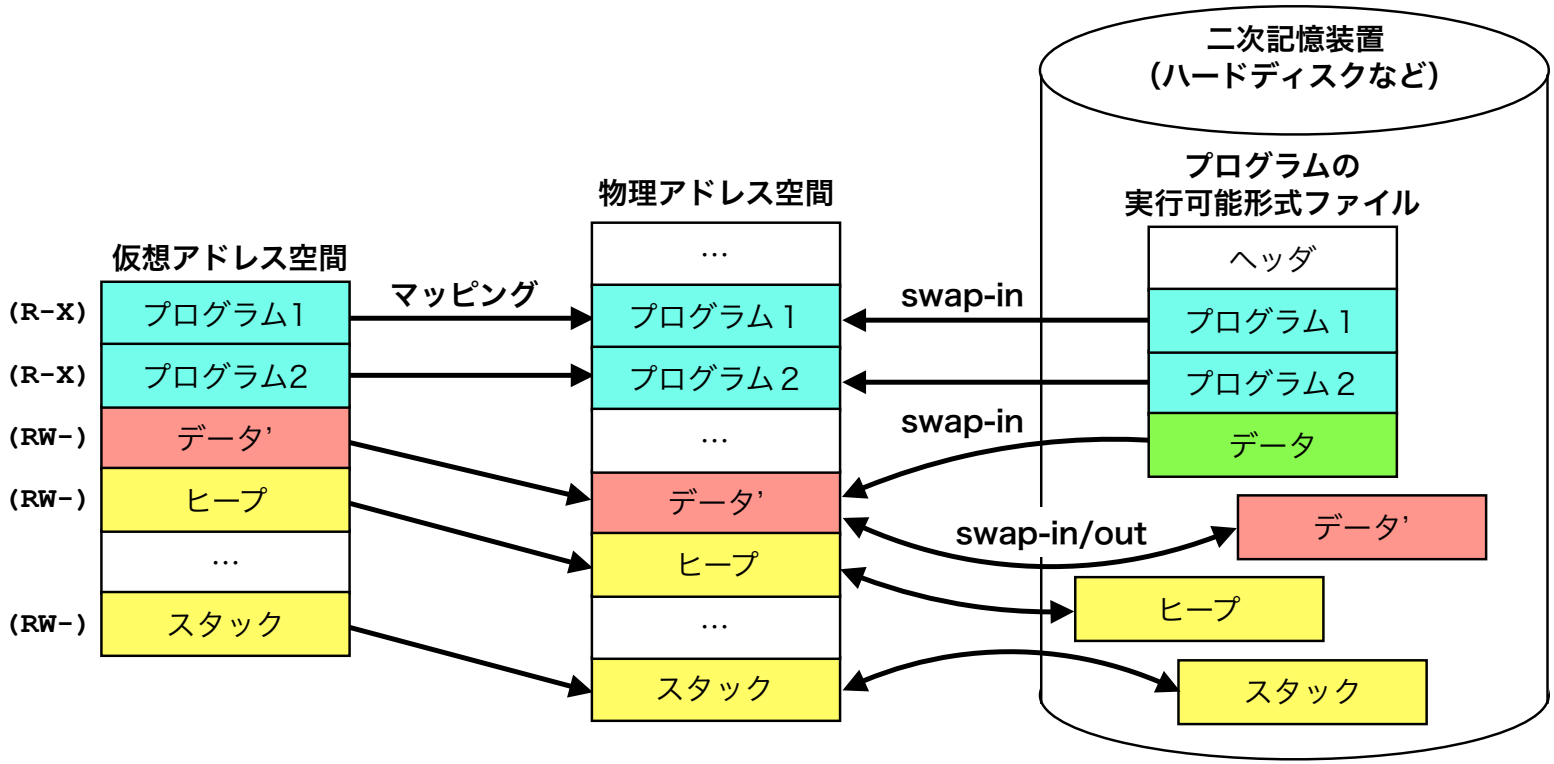

プロセスAの
仮想アドレス空間
(V=0) プログラムA1
(V=1) プログラムA2
(V=1) データA
(V=0) …
(V=1) スタックA
マッピング
物理アドレス空間
0 プログラムA2
1 データA
2 スタックA
3 プログラムB1
4 データB
5 スタックB1
swap-in
二次記憶装置
(ハードディスクなど)
プロセスAの
バッキングストア
プログラムA1
プログラムA2
データA
…
スタックA
プロセスBの
バッキングストア
プログラムB1
プログラムB2
データB
…
スタックB1
スタックB2
マッピング
プロセスBの
仮想アドレス空間
(V=1) プログラムB1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



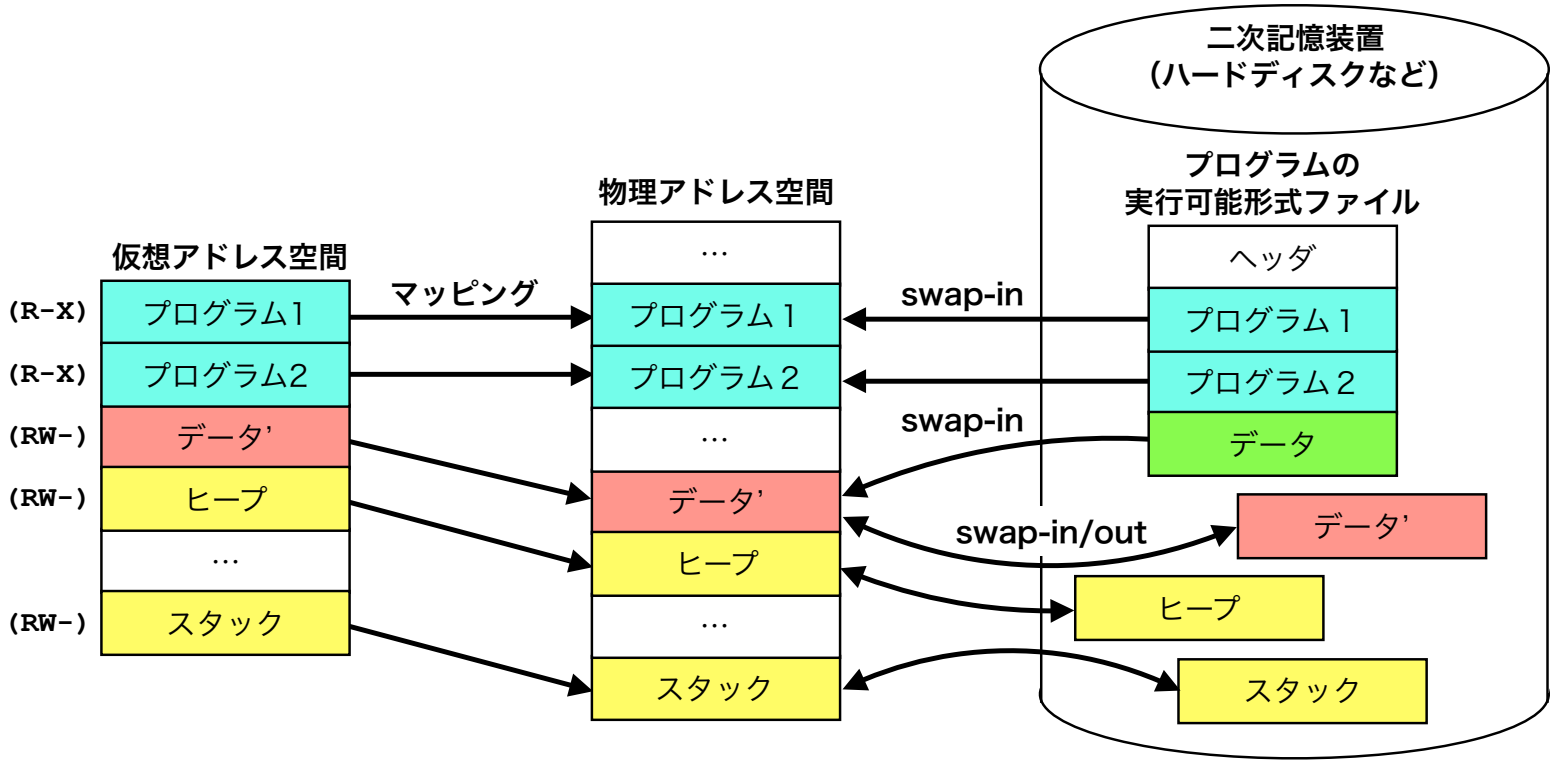
- ページを swap-in するための方式の一つ.
- 全てのページが不在の状態からスタートする.
- ページ不在を起こしたページを swap-in する.
  (使用しないページを読み込むような無駄が無い)

プログラムファイルの直接 swap-in による実行



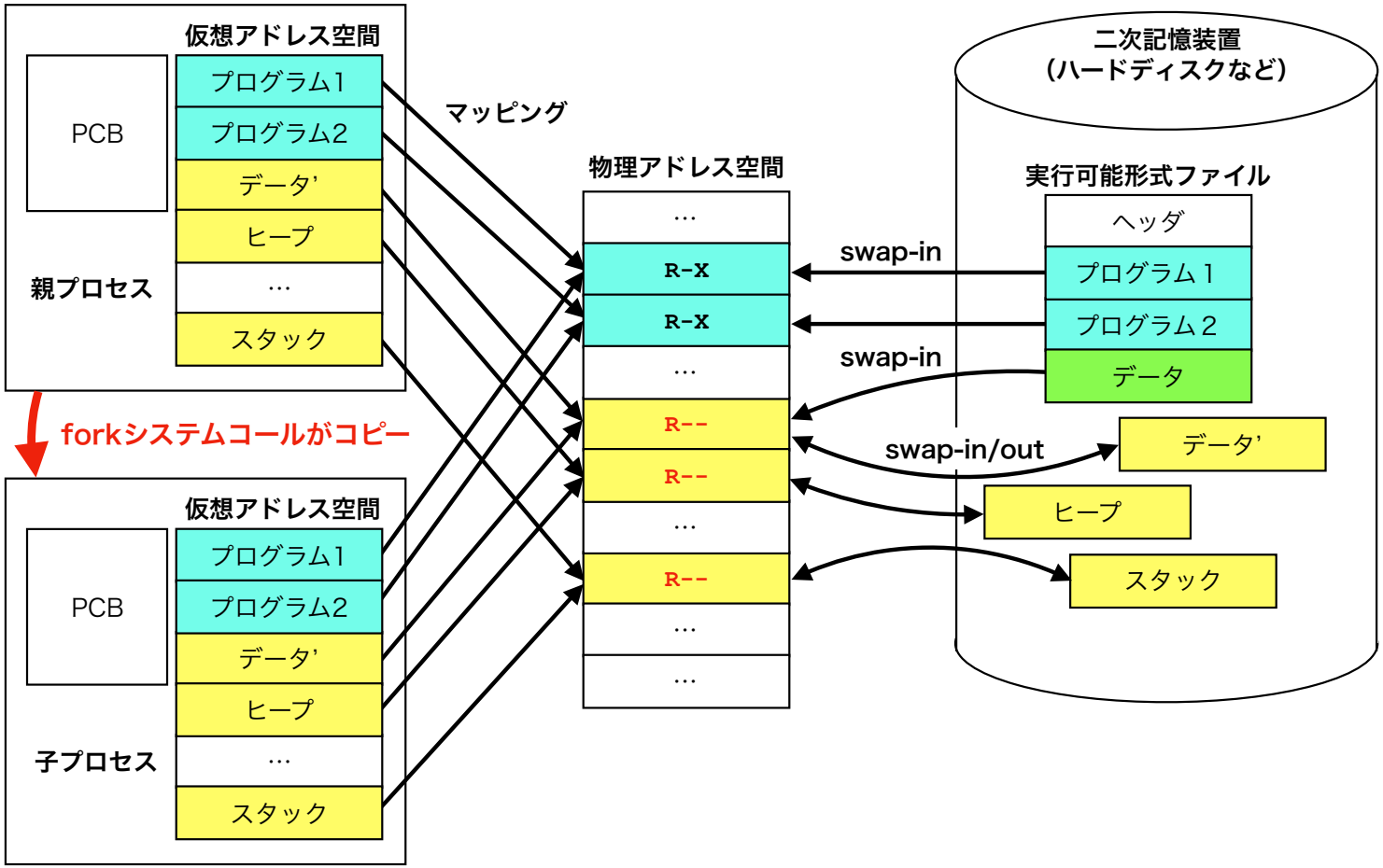
- デマンドページング用の実行可能形式ファイルを用いる．
  (このファイルはページサイズを意識した構造になっている)
- プログラムはファイルから swap-in する（R-X に設定)．
- 初期化データはファイルから swap-in する（RW-に設定)．
- 非初期化データ，ヒープ，スタックはゼロにする（RW-に設定)．

プログラムの swap-out



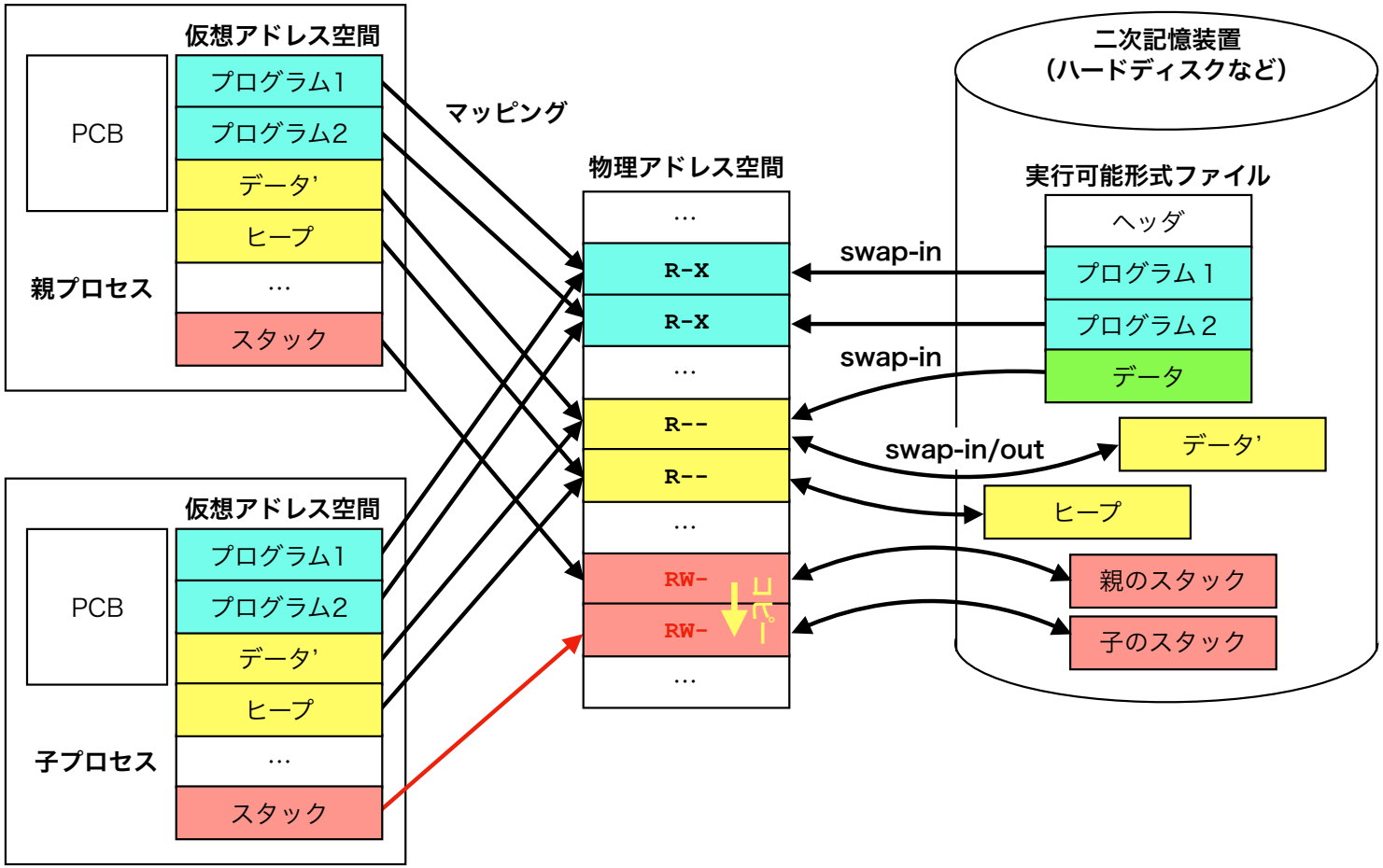
- フレームが枯渇したら使用頻度の低いフレームを解放し再利用する.
- プログラム（R-X）は変化しないので swap-out しない.
- 初期化データ（RW-）はバッキングストアに swap-out する.
- 非初期化データ，ヒープ，スタックも swap-out する.

Copy on Write（1）



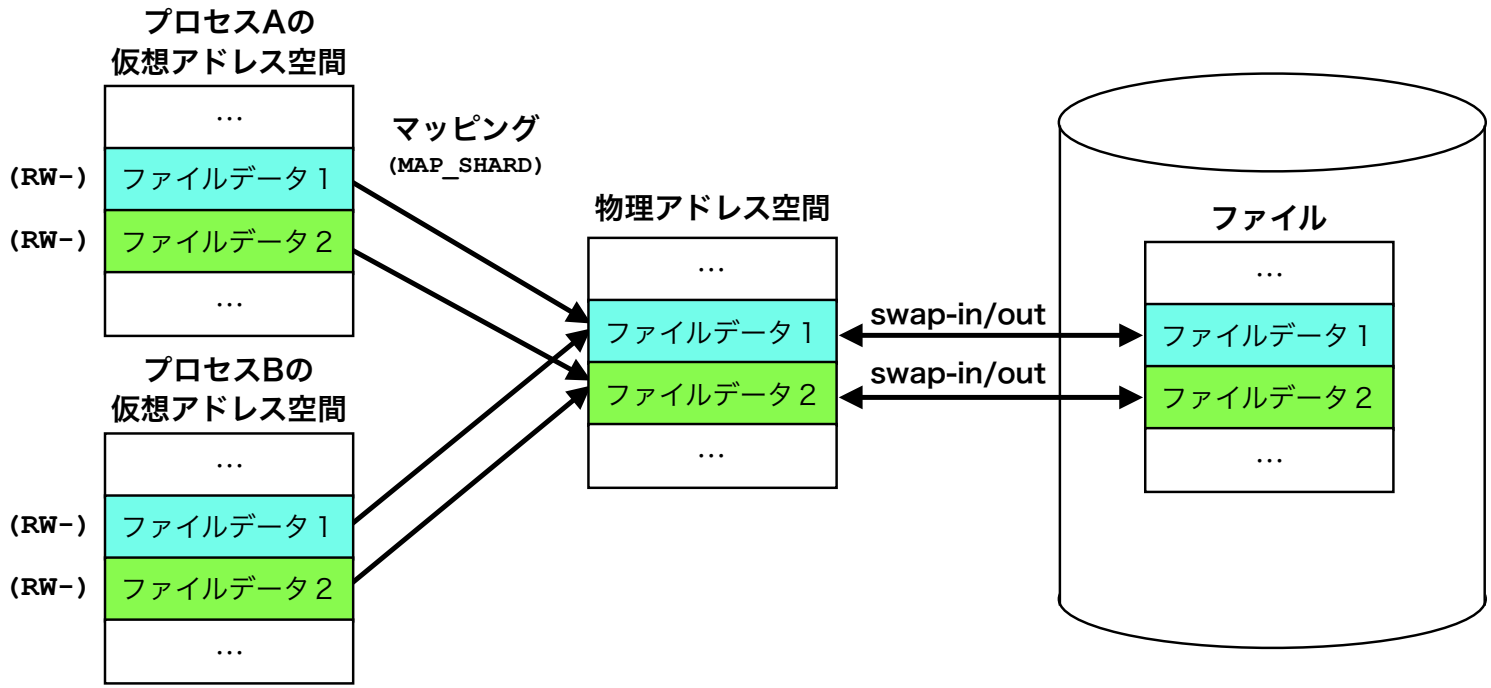
- fork-exec ではアドレス空間のコピーに無駄が多い．→ vfork
- vfork は使いにくい．使いやすい fork を改良する．
- fork の後，書き込み可能ページを一時的に R--に設定しておく．

Copy on Write（2）



- 例えばスタックに**書き込む**とメモリ保護違反割込みが発生する.
- 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.

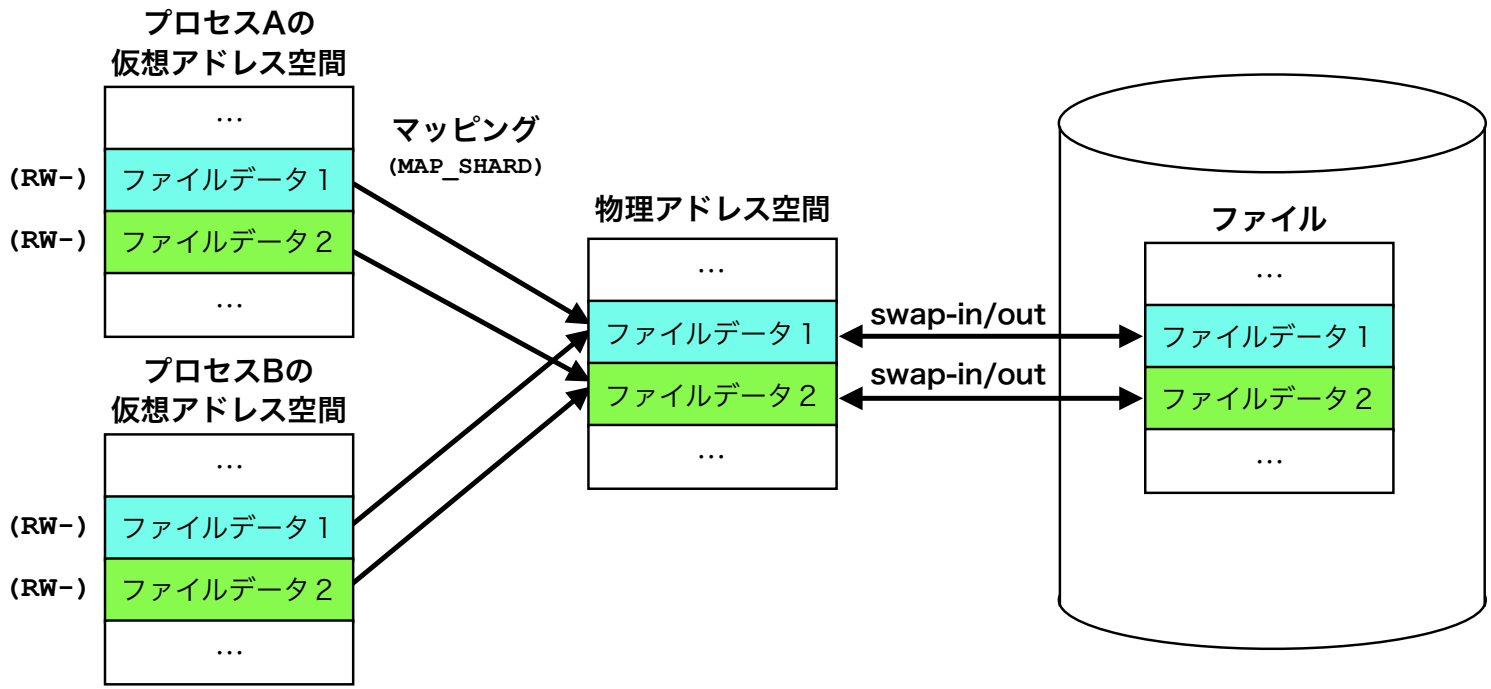
アドレスや長さはページサイズの整数倍にする.

メモリマップドファイル（3）

```
1  #include <stdio.h>                    // perror のために必要
2  #include <fcntl.h>                    // open のために必要
3  #include <unistd.h>                   // close のために必要
4  #include <sys/mman.h>                 // mmap のために必要
5  int main() {
6    int fd;
7    char *p, *fname="a.txt";
8    fd = open(fname, O_RDWR);           // 予め作成してある 4KiB のファイルを開く
9    if (fd<0) {
10     perror(fname);
11     return 1;
12   }
13   p = mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_FILE|MAP_SHARED,fd,0);
14   if (p==MAP_FAILED) {
15     perror("mmap");
16     return 1;
17   }
18   close(fd);                          // マップしたらクローズして良い
19   for (int i=0; i<4096; i++) {        // ファイルに A〜Z を繰り返し書き込む
20     p[i] = 'A' + (i % 26);
21   }
22   return 0;
23 }
```

メモリマップドファイル（4）

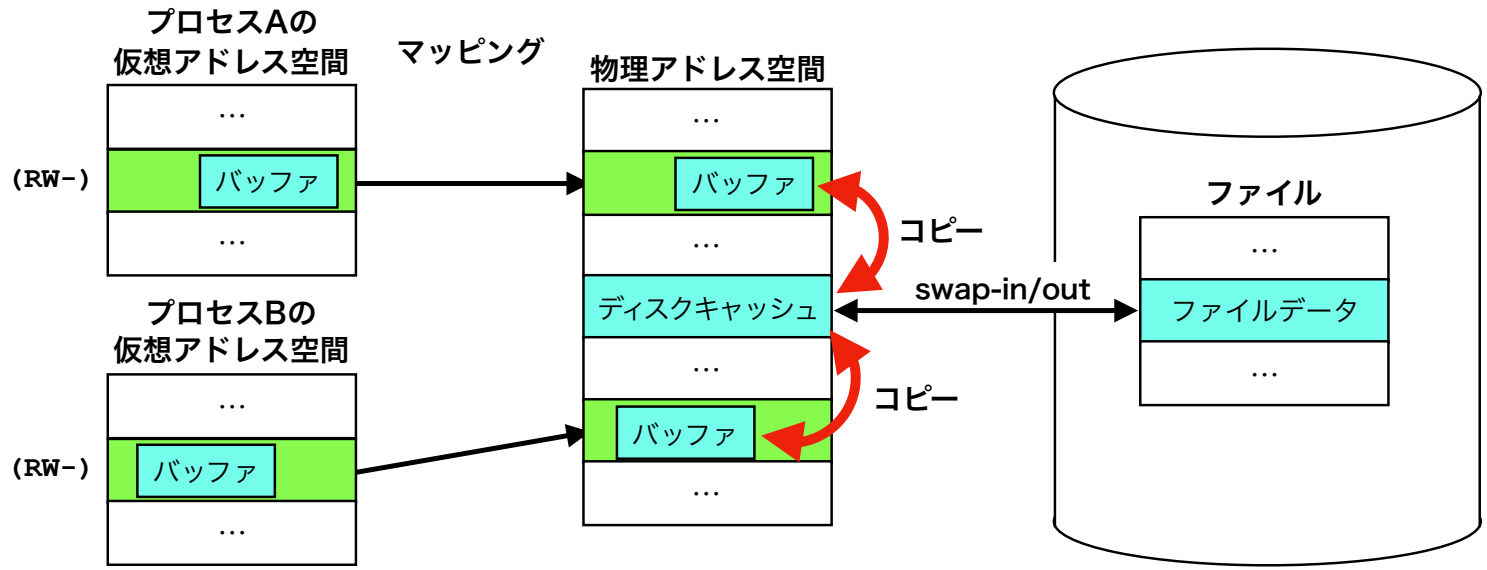
メモリマップドファイルの仕組み



- 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）

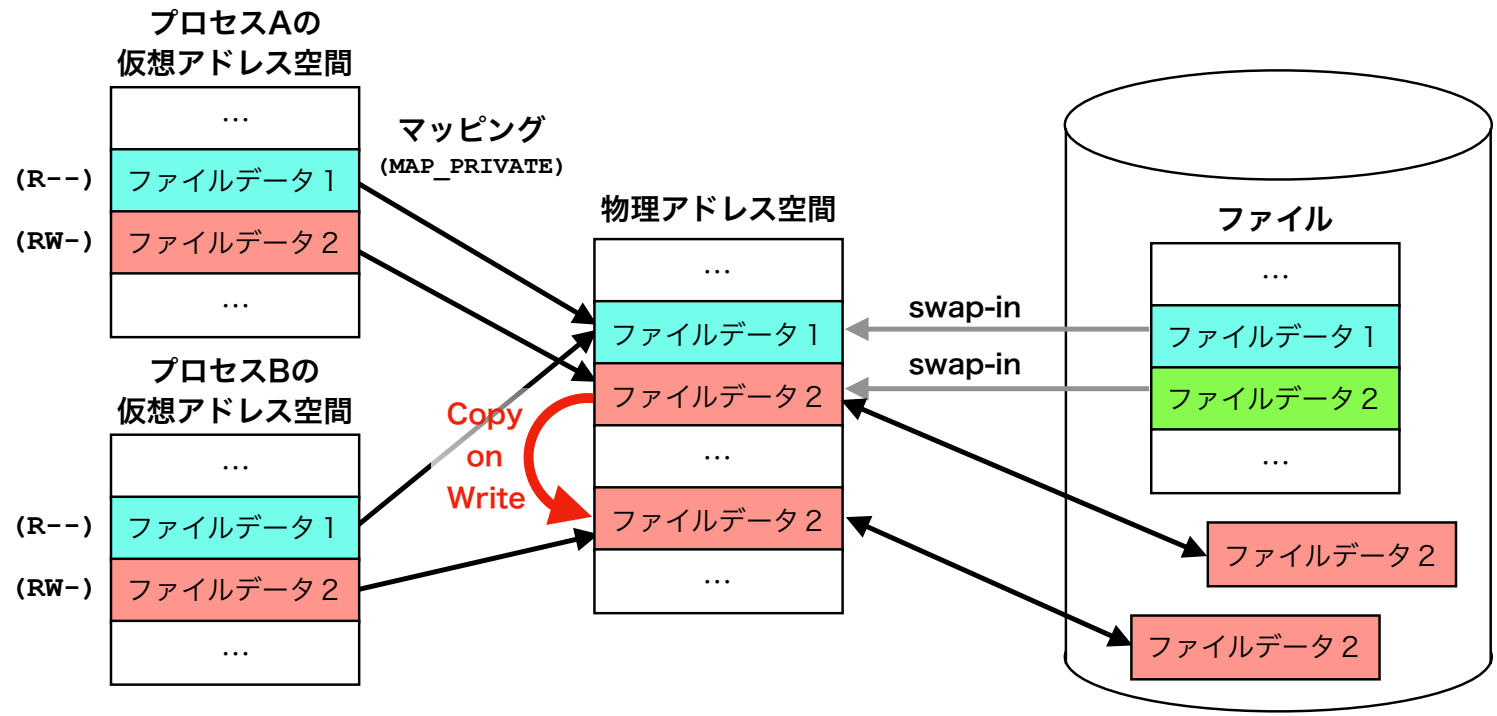
read/write システムコールとの比較



- 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）

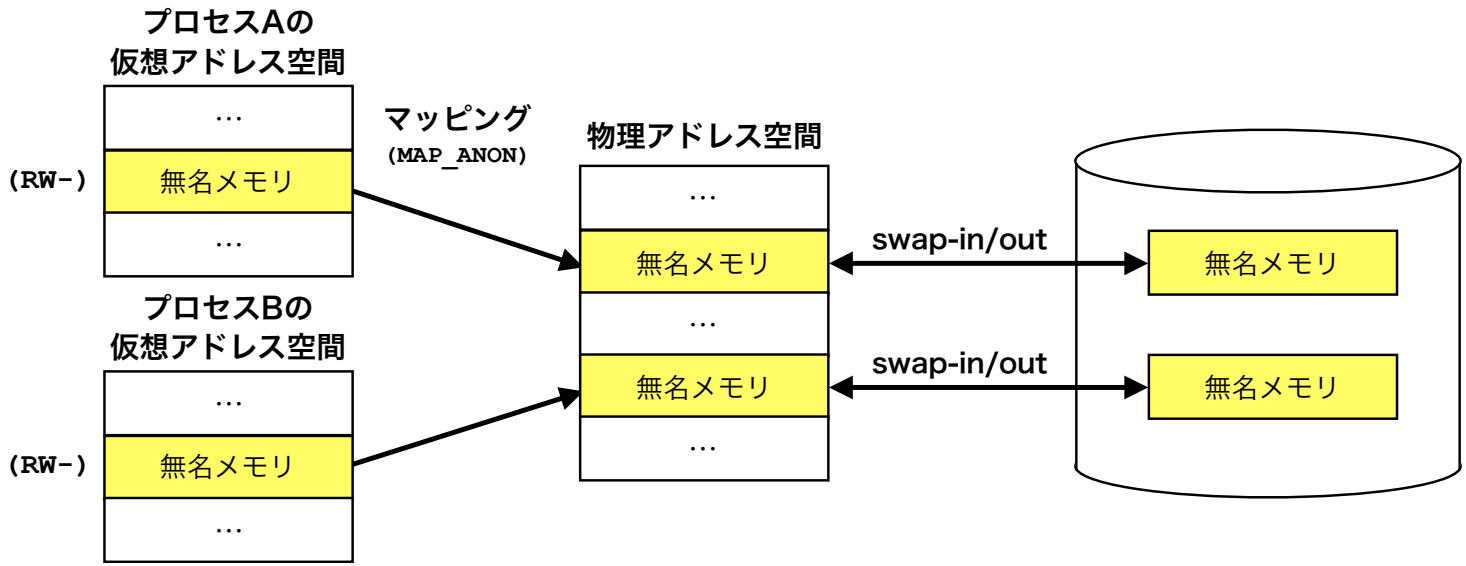
プロセスにローカルなマッピング



- これまでは MAP_SHARED 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）

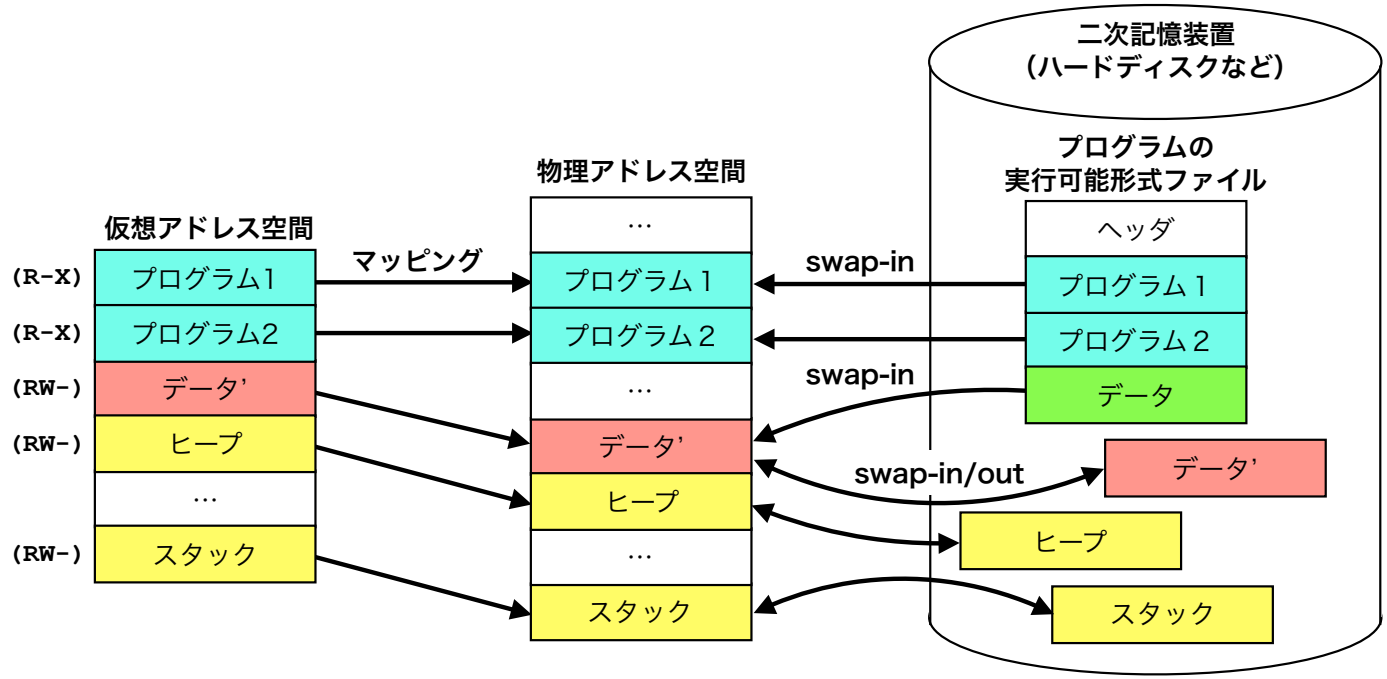
無名メモリ



- ファイルに関連付けられないがメモリ領域.
- MAP_ANON フラグを用いる.
- アクセスがあった時点で作成される（デマンドページング）.
- ページの初期値は**ゼロ**.

メモリマップドファイル（8）

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_SHARED/PRIVATE 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，スタックは無名メモリ（RW-，MAP_ANON）

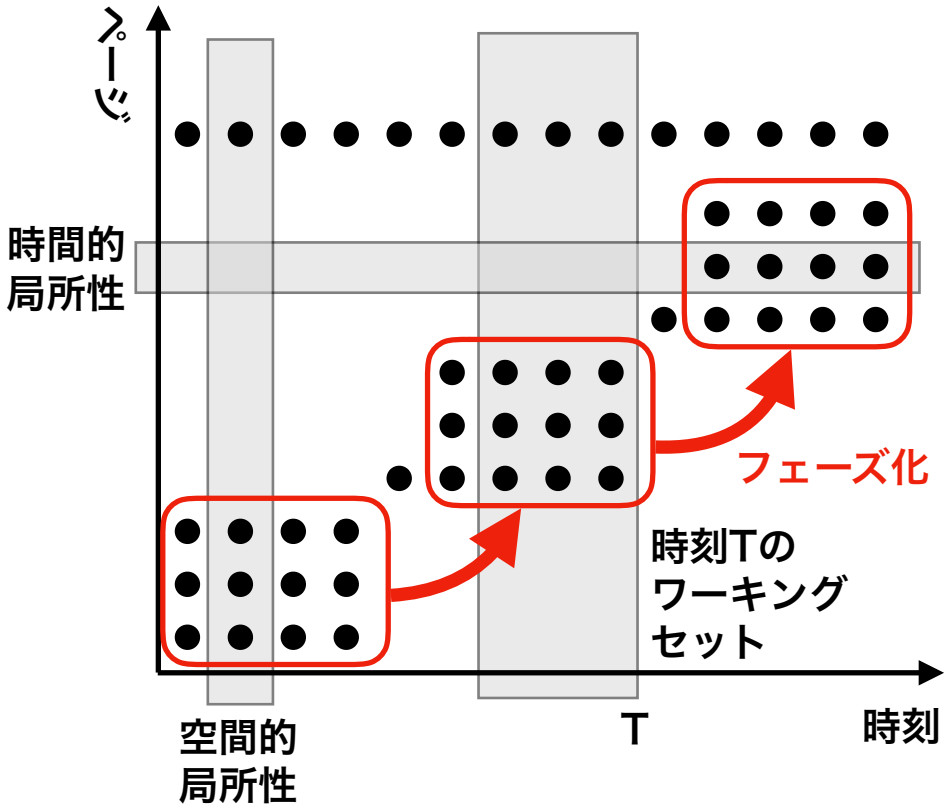
ページ置換えアルゴリズム

ページングによる仮想記憶で重要な三つのアルゴリズム

1. **ページ読み込みアルゴリズム**：いつページを swap-in するか決める．普通は，既に学んだデマンドページングを用いる．
2. **ページ置き換えアルゴリズム**：フレーム不足時に，どのページを再利用するか決める．
3. **フレーム割付けアルゴリズム**：どのフレームを使用するか決める．

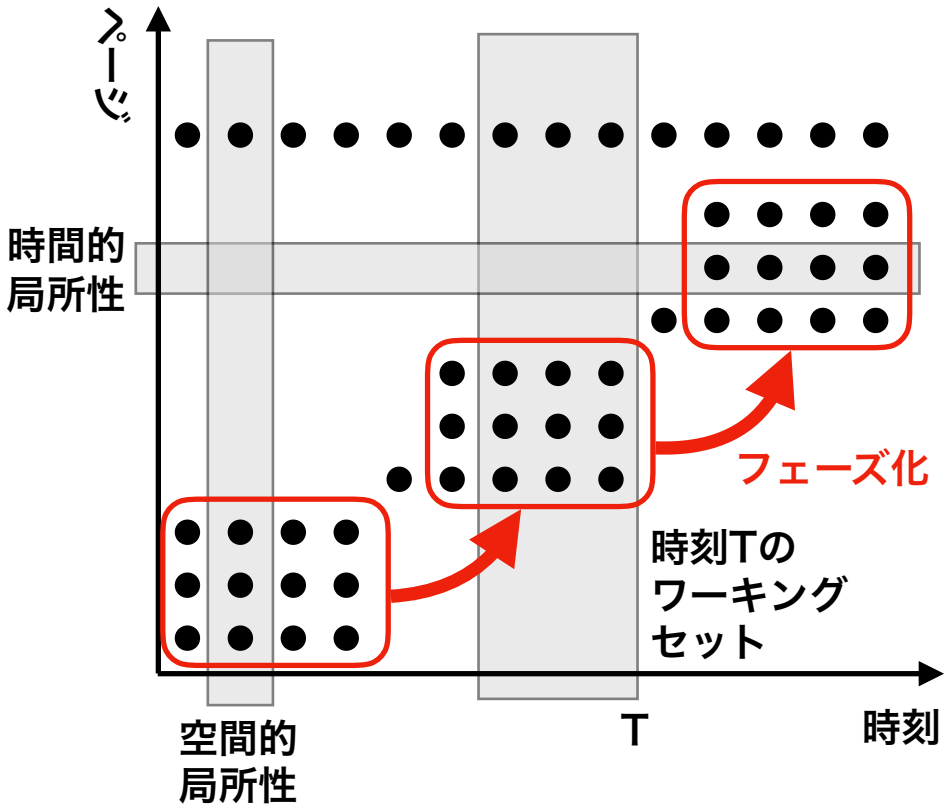
ページ置き換えアルゴリズムが，将来，使用されないフレームをうまく選択しないと，swap-out したページが直後に swap-in されることになり，システムの性能が著しく低下する．

局所性・ワーキングセット・フェーズ化（1）



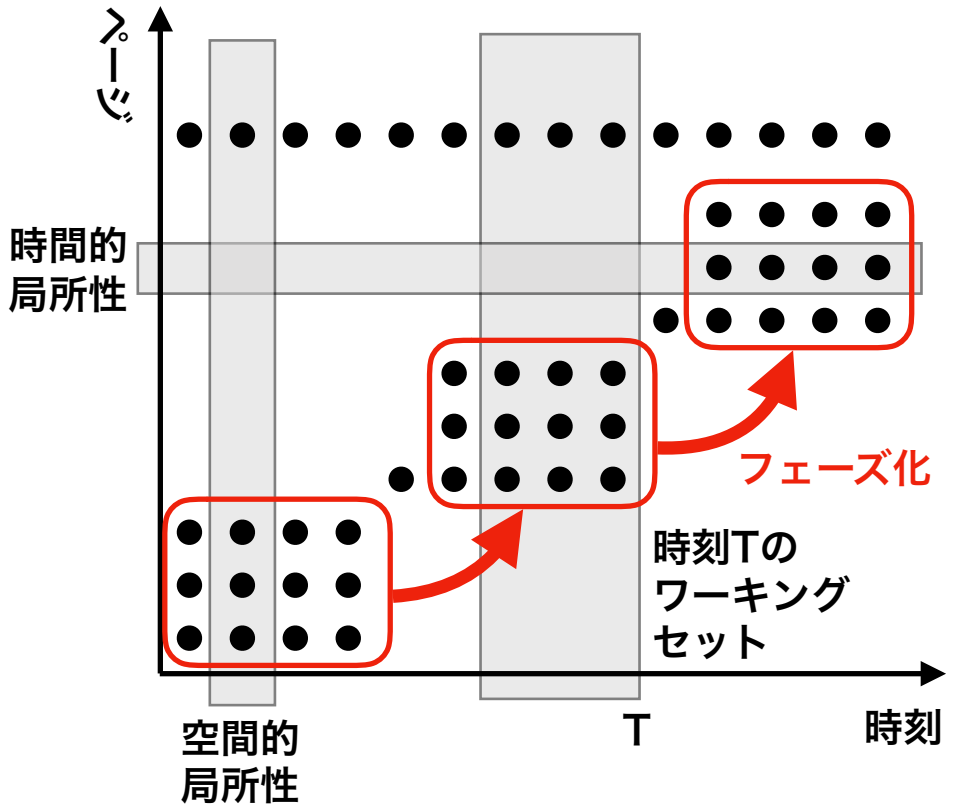
局所性 短い時間に着目すると，一部の連続したページが集中的にアクセスされる．→ **空間的局所性**
あるページに着目すると一部の連続した時刻にアクセスが集中している．→ **時間的局所性**

局所性・ワーキングセット・フェーズ化（2）



ワーキングセット　ある時間にアクセスされるページの集合のこと．
メモリに入り切らなくなると急激に性能が低下する．
→ **スラッシング**

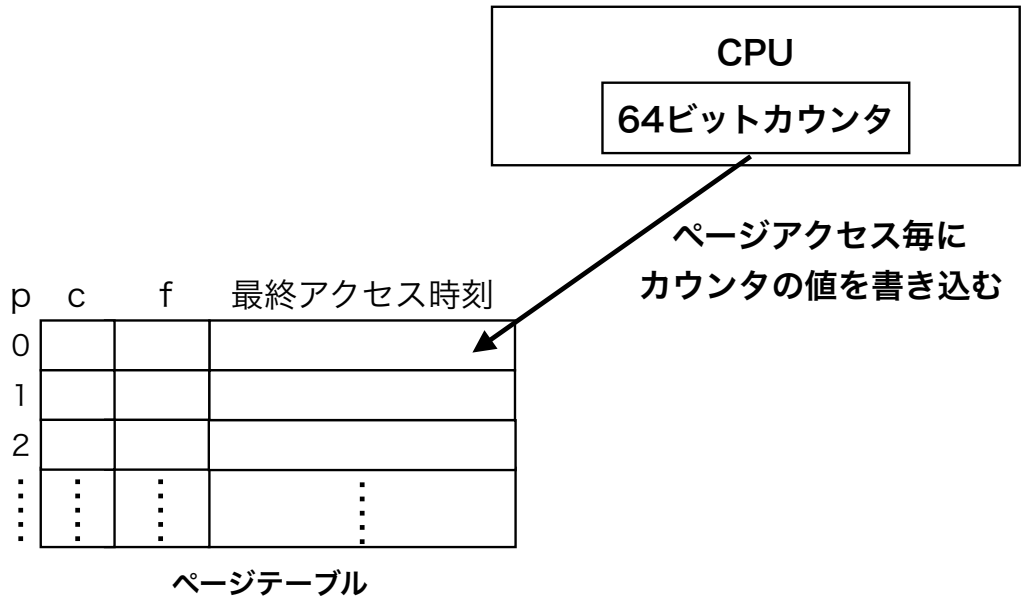
局所性・ワーキングセット・フェーズ化（3）



フェーズ化現象 ワーキングセットが急激に変化する現象のこと．
「入力フェーズ」，「計算フェーズ」，「出力フェーズ」
フェーズ遷移時は局所性が失われる → ページ不在集中

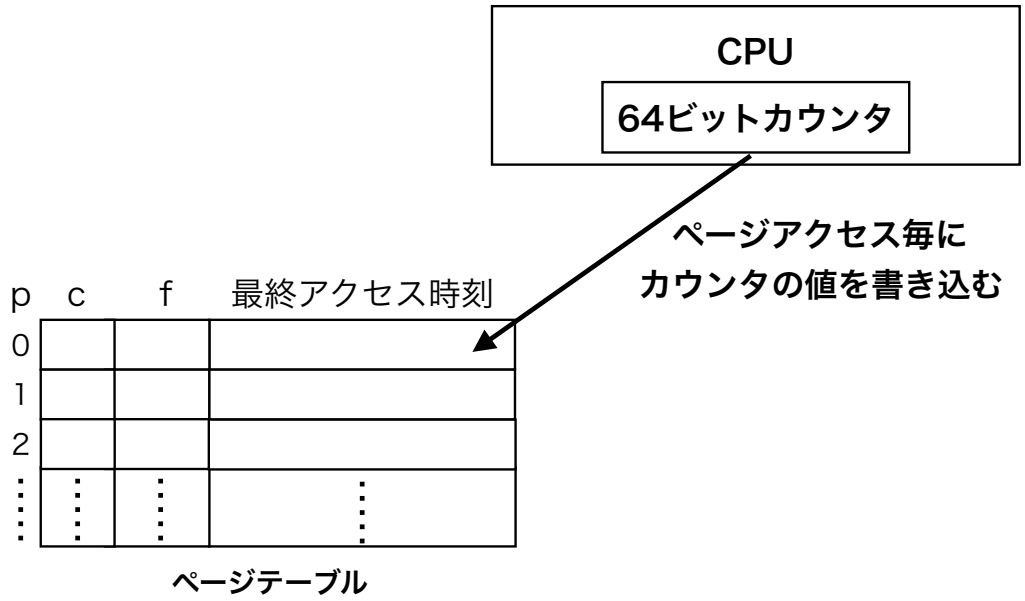
LRU（Least Recently Used）アルゴリズム（1）

仮定：最近アクセスされていないページは，この先，アクセスされない．
(時間的局所性があるなら最良な方法．)



1. メモリアクセス毎にページテーブルにカウンタの値を書く．
2. ページテーブルをスキャンし最も古いページを見つける．
3. 見つけたページを swap-out し目的のページを swap-in する．（置換え）

LRU（Least Recently Used）アルゴリズム（2）



問題点（LRU の完全な実装は困難と言われている）

1. ハードウェアのコスト
2. ページ不在時の処理の重さ．（ページ不在は頻繁に発生）

macOS の vm_stat で調べると毎秒数千回のページ不在！！

LFU（Least Frequently Used）アルゴリズム

NFU（Not Frequently Used）とも呼ばれる.

- LRU の近似方式の一種である.
- ページテーブルの R ビットを使用.
- フレーム毎にカウンタを準備.
- 特別なハードウェアは不要.

アルゴリズム

1. R ビットとフレームのカウンタをゼロにクリアする.
2. 定期的（例えば TICK=20ms 毎）にページテーブルをスキャンする. R=1 のエントリを見つけたら対応するフレームのカウンタをインクリメントし，R をゼロにクリアする.
3. ページ不在時にフレームが不足したなら，カウンタの値が最小のフレームを置き換える.

エージングアルゴリズム

LFU（Least Frequently Used）アルゴリズムの改良

LFU の問題点
一度カウンタの値が大きくなると，使用されなくなっても置き換えが起こらない．

LFU の改良
定期的にページテーブルをスキャンする際のカウンタの更新方法を次のように改良する．

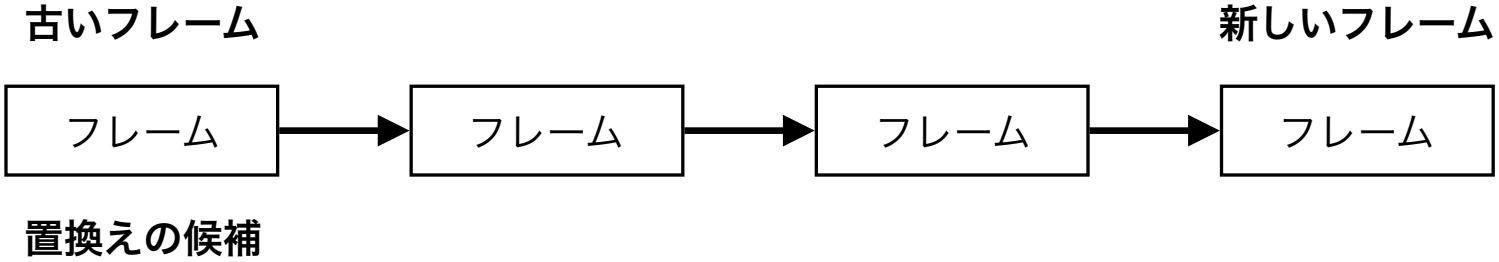
R=1 のフレーム $cnt \leftarrow cnt \div 2 + 0x8000$（カウンタは 16bit と仮定）

R=0 のフレーム $cnt \leftarrow cnt \div 2$

この改良により，過去の R ビットの影響が徐々に小さくなる．

FIFO（First-In First-Out）アルゴリズム

仮定：「長くメモリに滞在しているページは役割を終えている」
特別なハードウェアを用いることなく，ソフトウェアだけで実現できる．



アルゴリズム

1. swap-in したフレームをリストの最後に追加する．
2. フレームが不足時は，リストの先頭のフレームを置き換える．

ページテーブルのスキャンが不要なので非常に軽い．
常時使用されるページも時間が経過すると swap-out される問題がある．
Belady **の異常な振る舞い**をすることがある．

Beladyの異常な振る舞いの例

FIFOアルゴリズムを用い,
ページ参照ストリング（W：1 2 3 4 1 2 5 1 2 3 4 5）の場合

- フレーム数（m=3）の場合（ページ不在9回）

W	1	2	3	4	1	2	5	1	2	3	4	5
S	*1	*2	*3	*4	*1	*2	*5	5	5	*3	*4	4
		1	2	3	4	1	2	2	2	5	3	3
			1	2	3	4	1	1	1	2	5	5

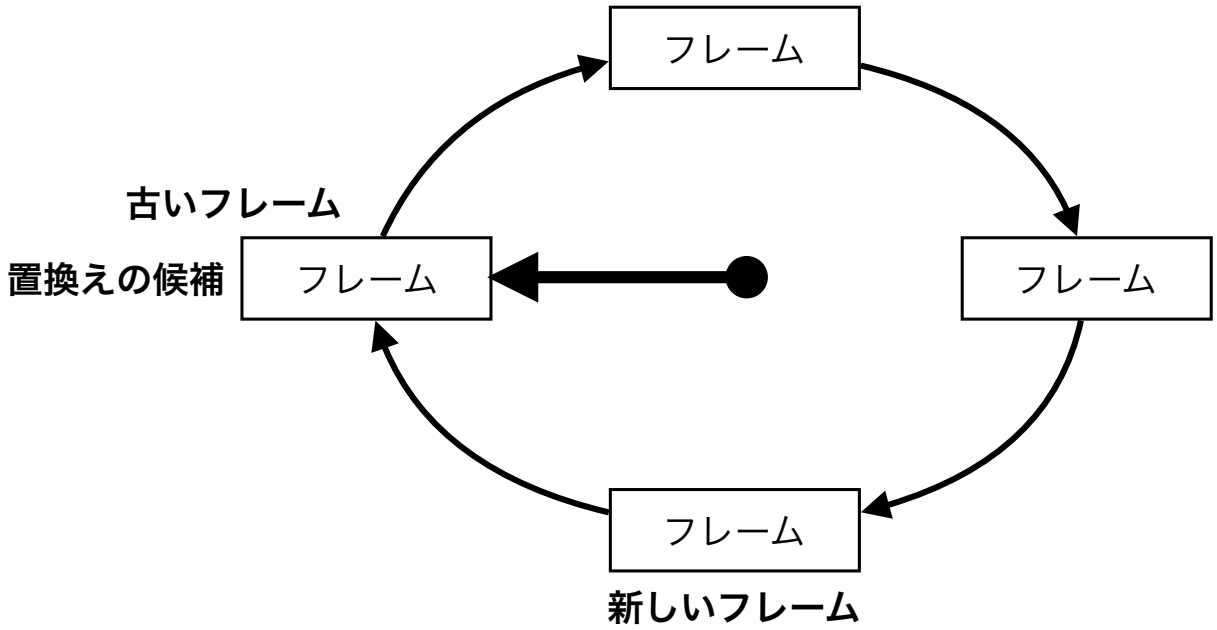
- フレーム数（m=4）の場合（ページ不在10回）

W	1	2	3	4	1	2	5	1	2	3	4	5
S	*1	*2	*3	*4	4	4	*5	*1	*2	*3	*4	*5
		1	2	3	3	3	4	5	1	2	3	4
			1	2	2	2	3	4	5	1	2	3
				1	1	1	2	3	4	5	1	2

メモリが多い方（m=4）のページ不在回数が多い.

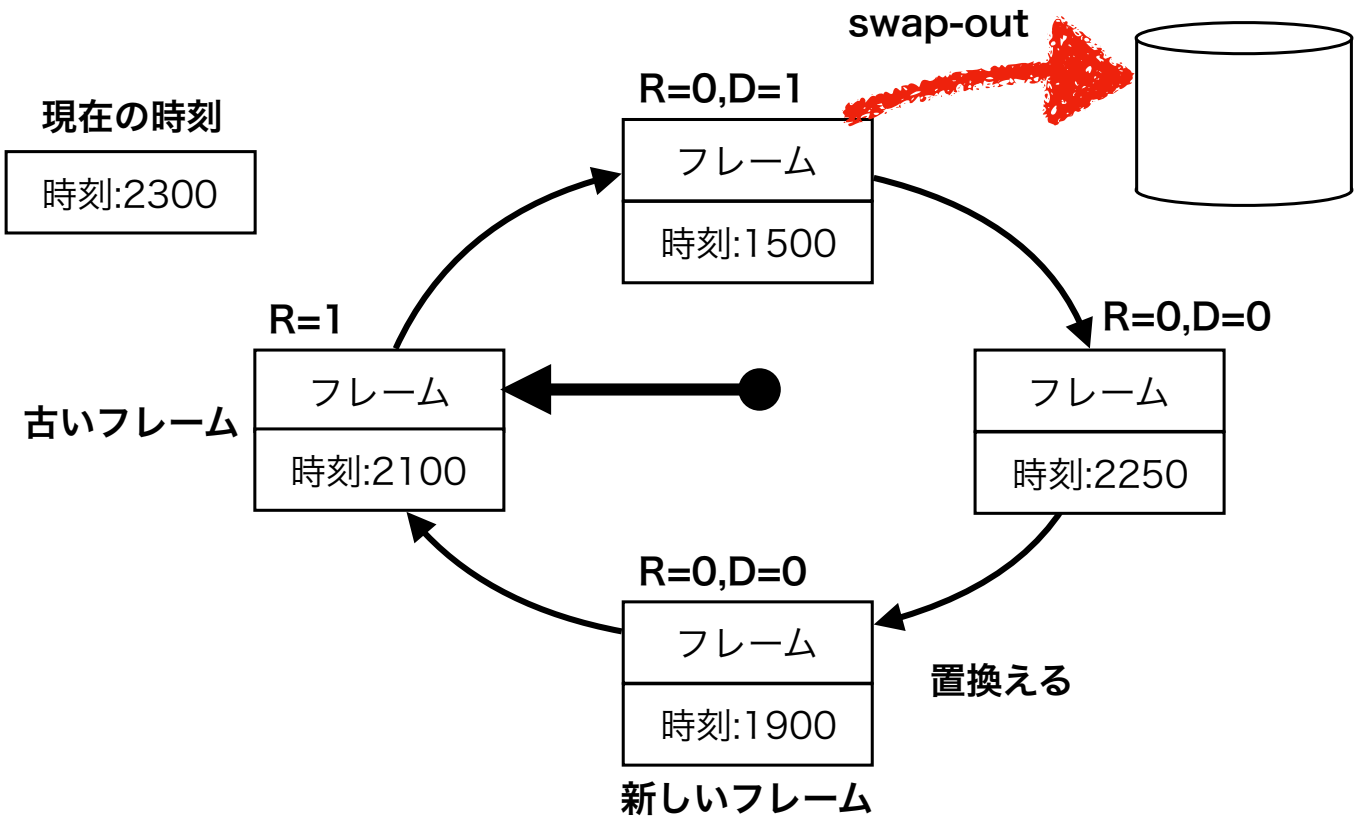
Clock アルゴリズム

環状リストを用いる．R ビットも使用する．



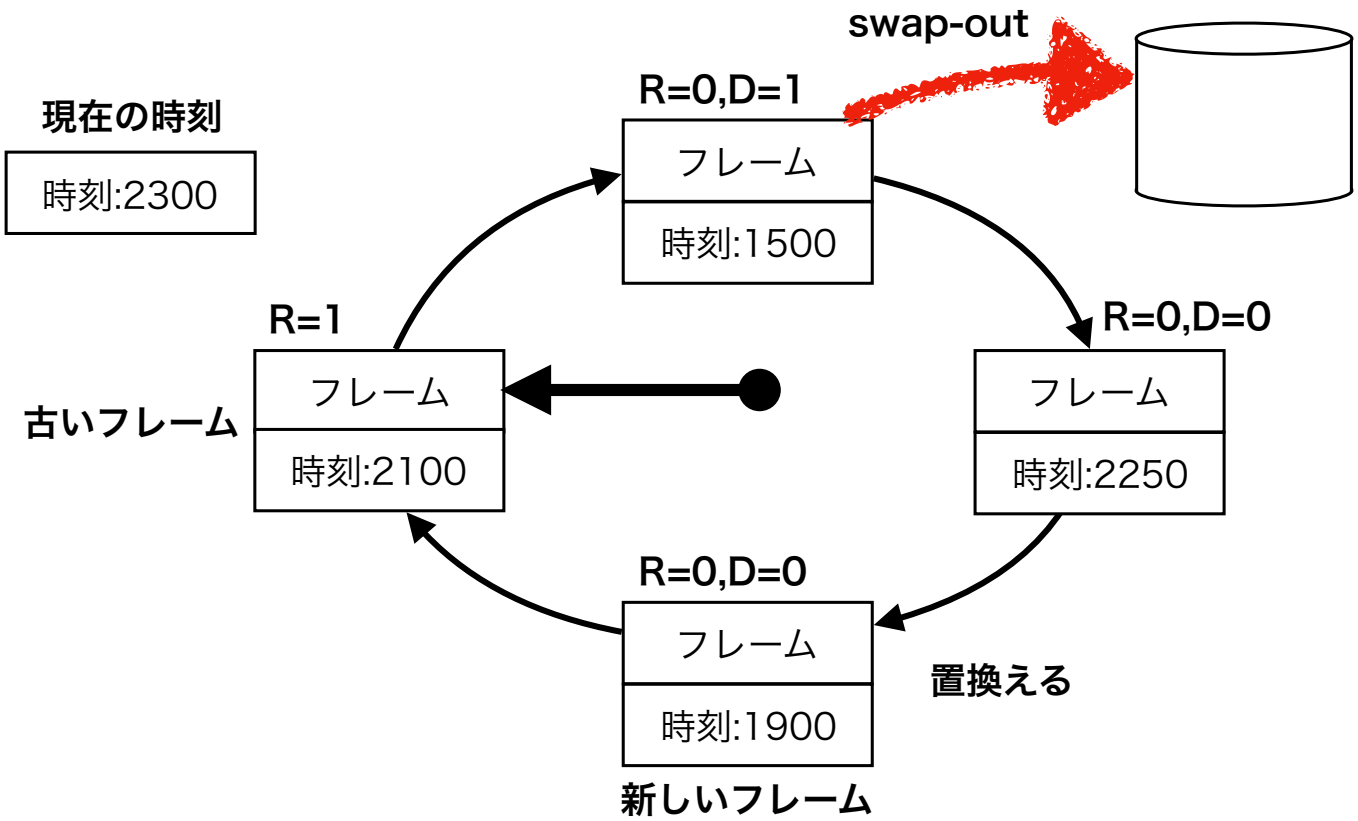
1. swap-in する度にフレームを環状リストに挿入していく．
2. 定期的（例えば TICK=20ms 毎）に R ビットをクリアする．
3. 時計の針が指しているフレームの R ビットを調べる．
   R=0 の場合 ページは古い＋最近アクセスされていない．→ 置換え
   R=1 の場合 ページは最近アクセスされている．→ 針を進める
   最悪でも時計の針が一周回ると R=0 のページが見つかる．

WSClock アルゴリズム（1）



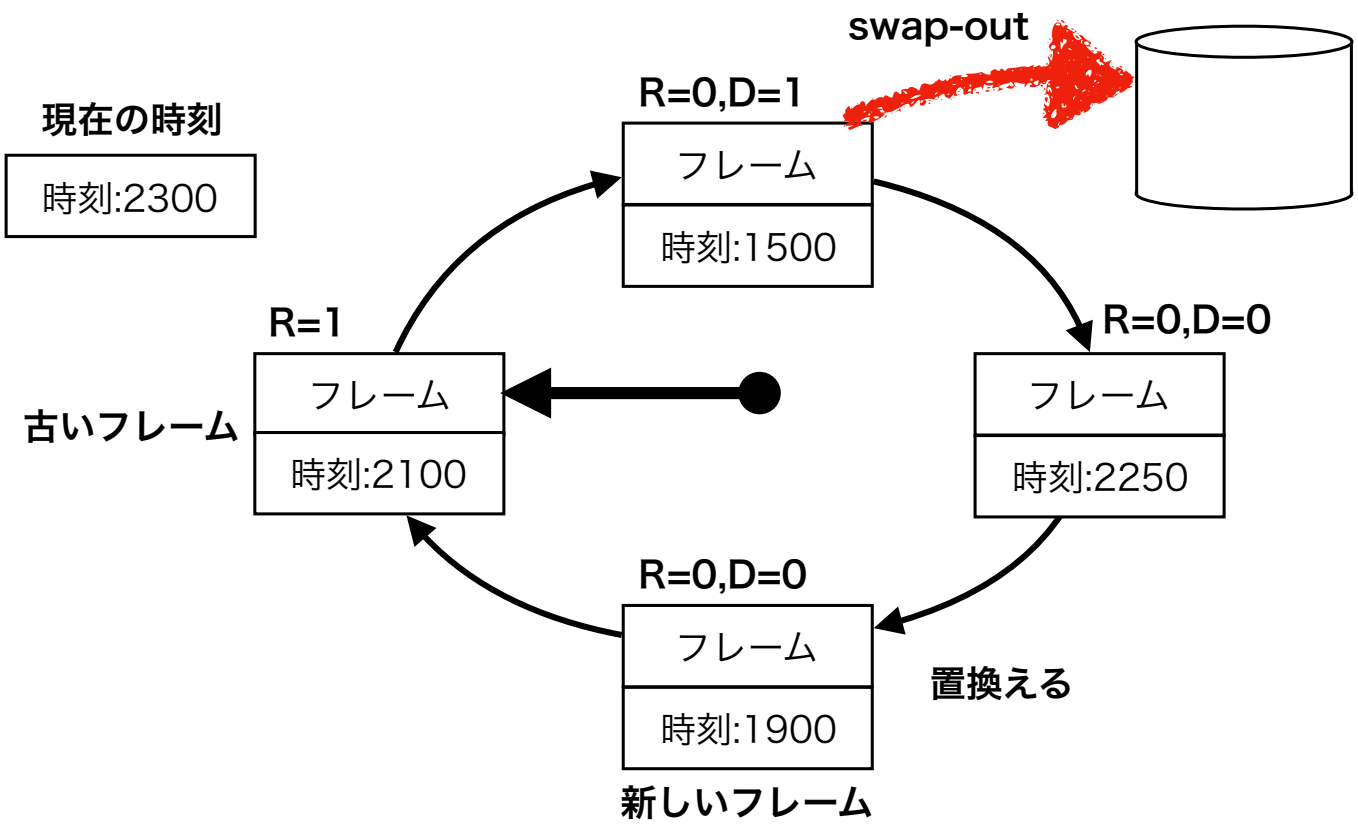
- ワーキングセットを考慮した Clock アルゴリズムである.
- 単純でパフォーマンスが良いので広く使用されている.
- アクセス時刻を記録した環状リストに用いる.

WSClock アルゴリズム（2）



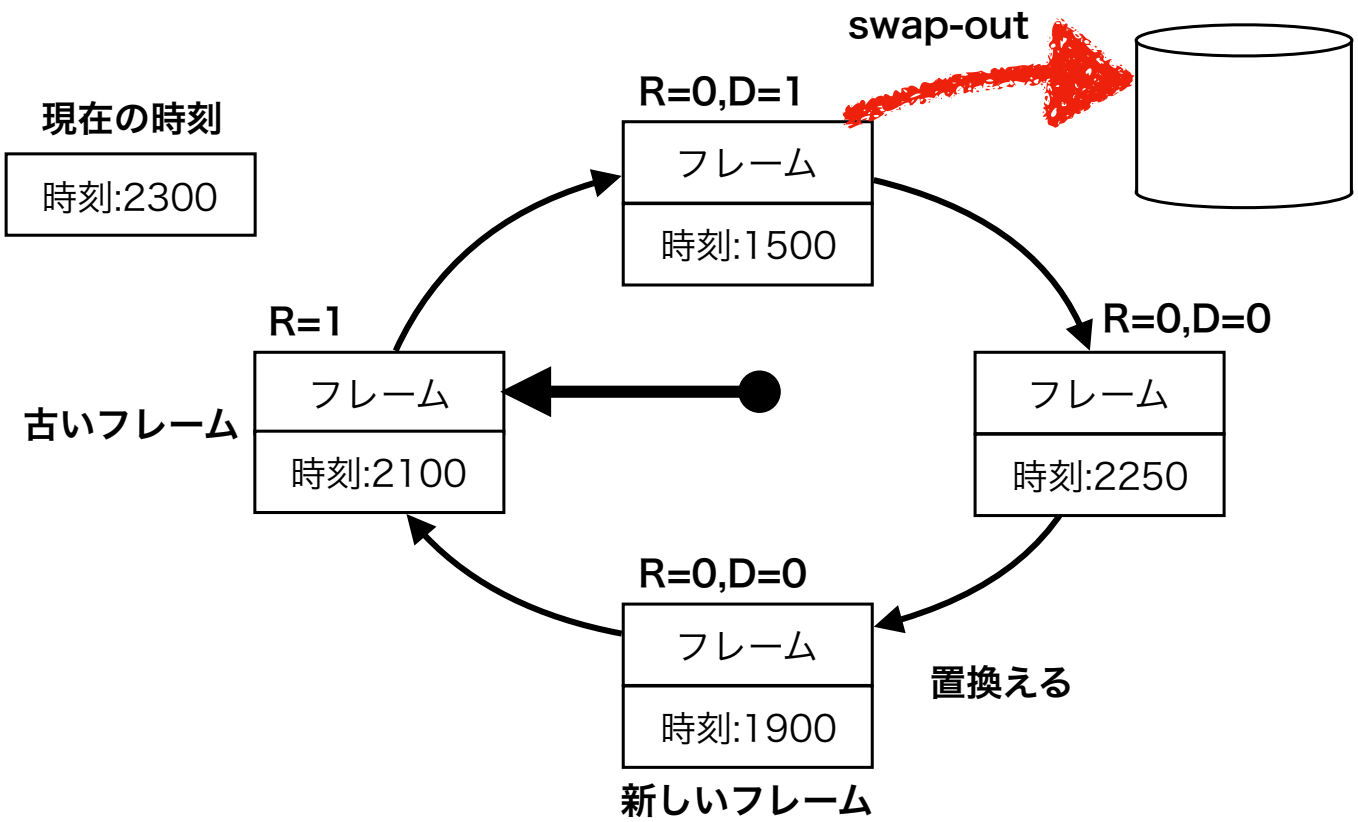
- 時刻が古くなっているフレームはワーキングセット外と判断.
- ページテーブルの R ビットと D ビットも使用する.

WSClock アルゴリズム（3）



1. swap-in する度にフレームを環状リストに挿入していく．
2. 定期的に全テーブルエントリの R ビットをクリアする．
   その際，R=1 だったフレームだけに現在時刻を記録する．

WSClock アルゴリズム（4）



3. フレーム不足なら時計の針のフレームを調べる.

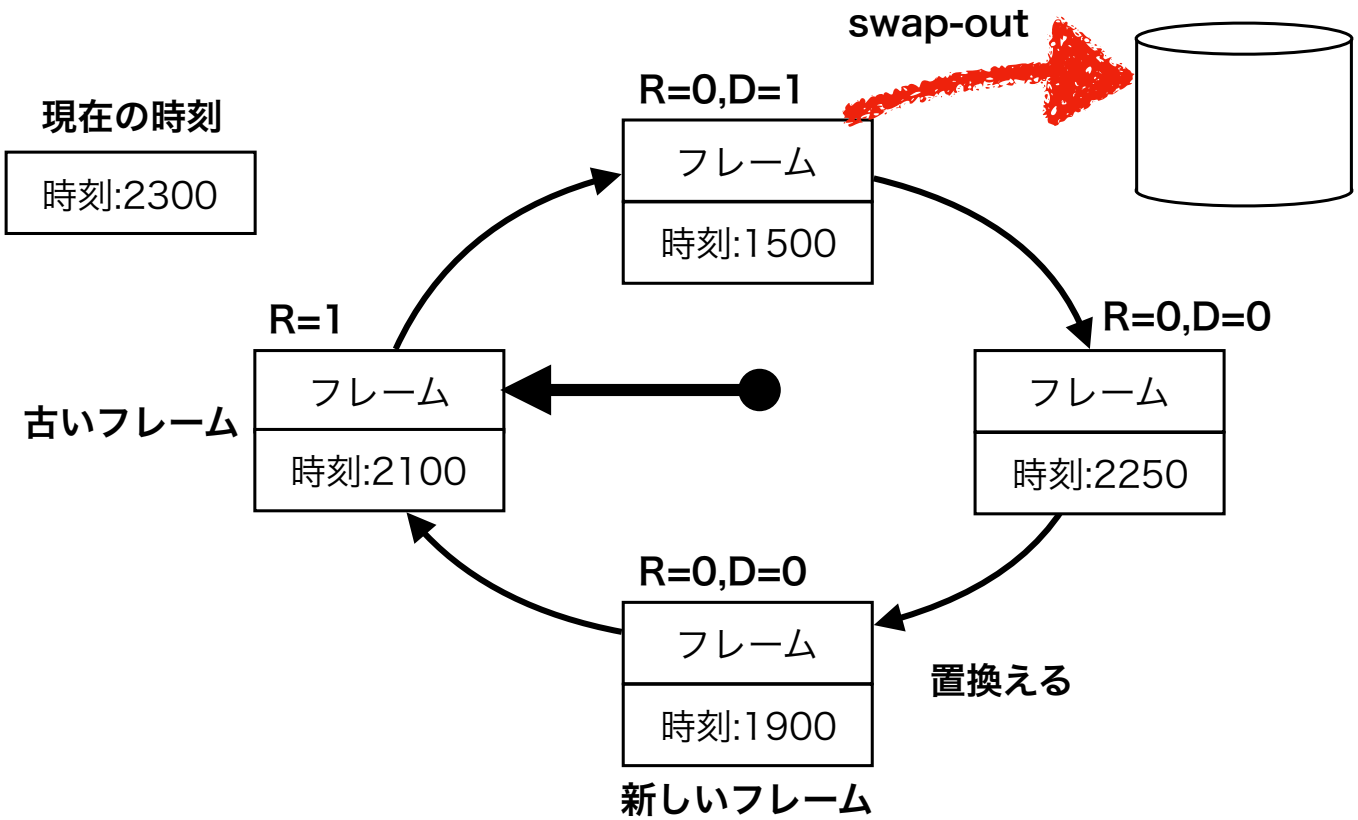
   R=1 の場合 R ビットをクリアして次のフレームに進む.

   時刻が新しい場合 次のフレームに進む.

   時刻が古い場合 ページはワーキングセットに含まれていない.

   ページの置換え処理を行う.

WSClock アルゴリズム（5）



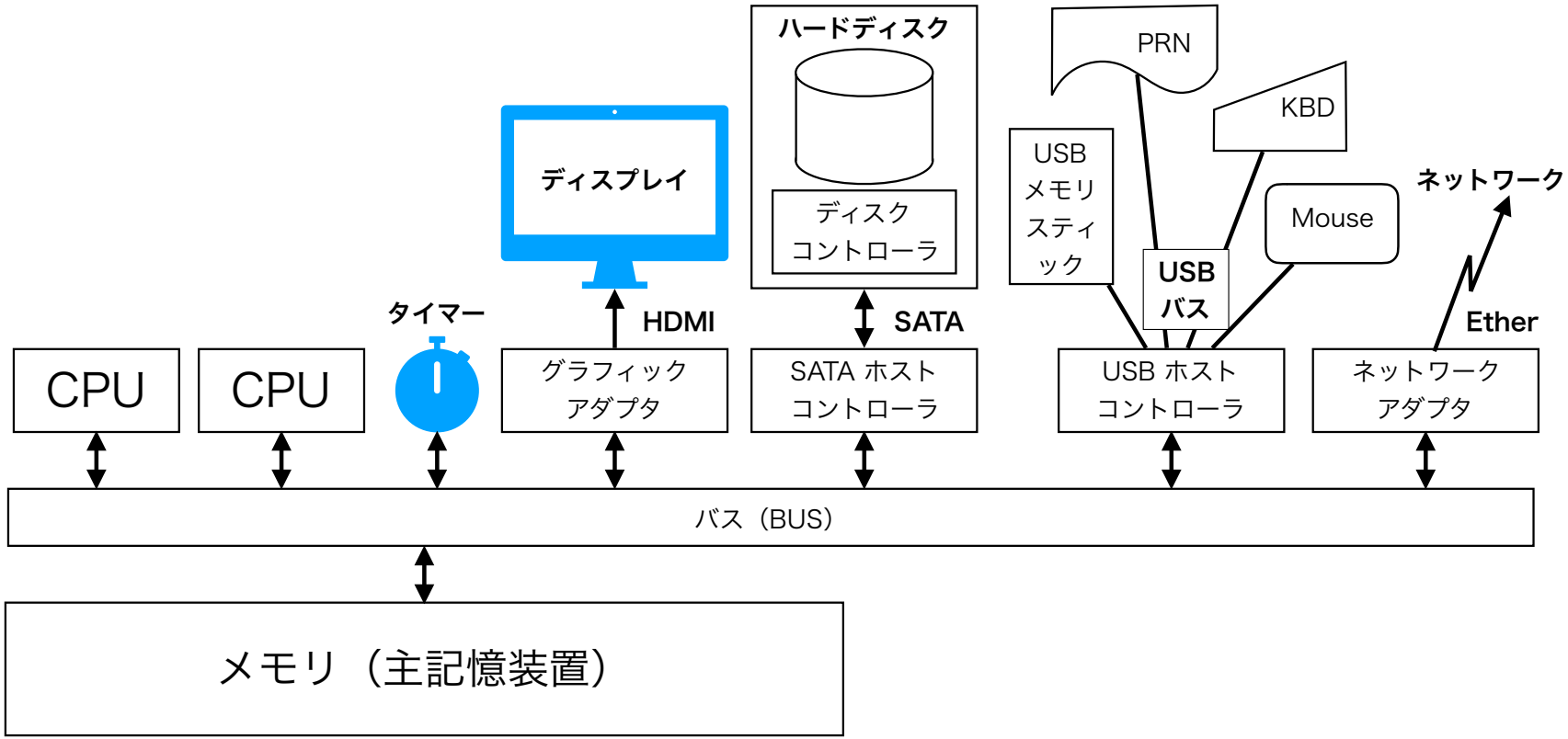
3. フレーム不足なら時計の針のフレームを調べる.

   時刻が古い場合 **ページの置換え処理**を行う.

   D=1 の場合 swap-out を予約し次のフレームに進む.

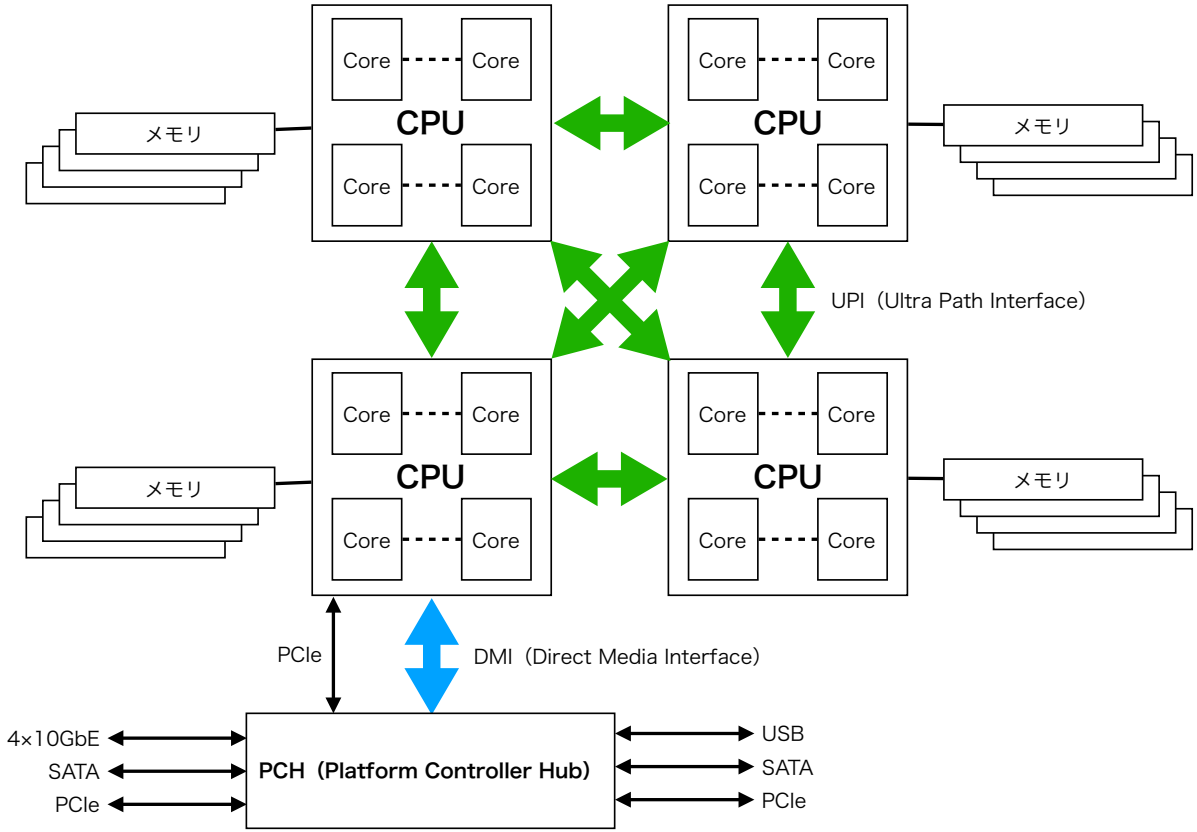
   D=0 の場合 このフレームを置き換える.

フレーム割り付けアルゴリズム（1）



- フレームはどれも均質
- SMP システムであってもそうである.

フレーム割り付けアルゴリズム（2）



- 近年のサーバ用 SMP では均質ではない.
- メモリと CPU からなるノードを相互接続している.
- 自ノードと他ノードでメモリアクセス時間が異なる.
- このことを考慮したフレーム割付と CPU スケジューリングが必要.

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
   - 局所性
   - ワーキングセット
   - フェーズ化
   - スラッシング
   - ページ読み込みアルゴリズム
   - ページ置き換えアルゴリズム
   - ページ割付けアルゴリズム
   - Belady の異常な振る舞い

練習問題

2. 「Belady の異常な振る舞いの例」で示したページ参照ストリングとフレーム数を用い，他のページ置き換えアルゴリズムを適用した場合をトレースしなさい．