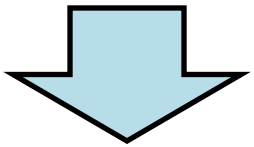
6章　仮想記憶
（Virtual Memory）

仮想記憶とは？

- ユーザから見えるメモリをハードウェアとは独立させて，プログラムを実行させる技術

- 空間の大きさについて
  - ➢今までの話
    - ✓プロセス全体はメモリ上になければならなかった.



  - ➢仮想記憶では
    - ✓必ずしもメモリ上に全部ある必要なし.

デマンドページング（Demand paging，要求ページング）

- ページングを用いる．
- プログラム実行時に必要になったら，そのページを2次記憶（ディスク）からメモリにもってくる（格納する）．

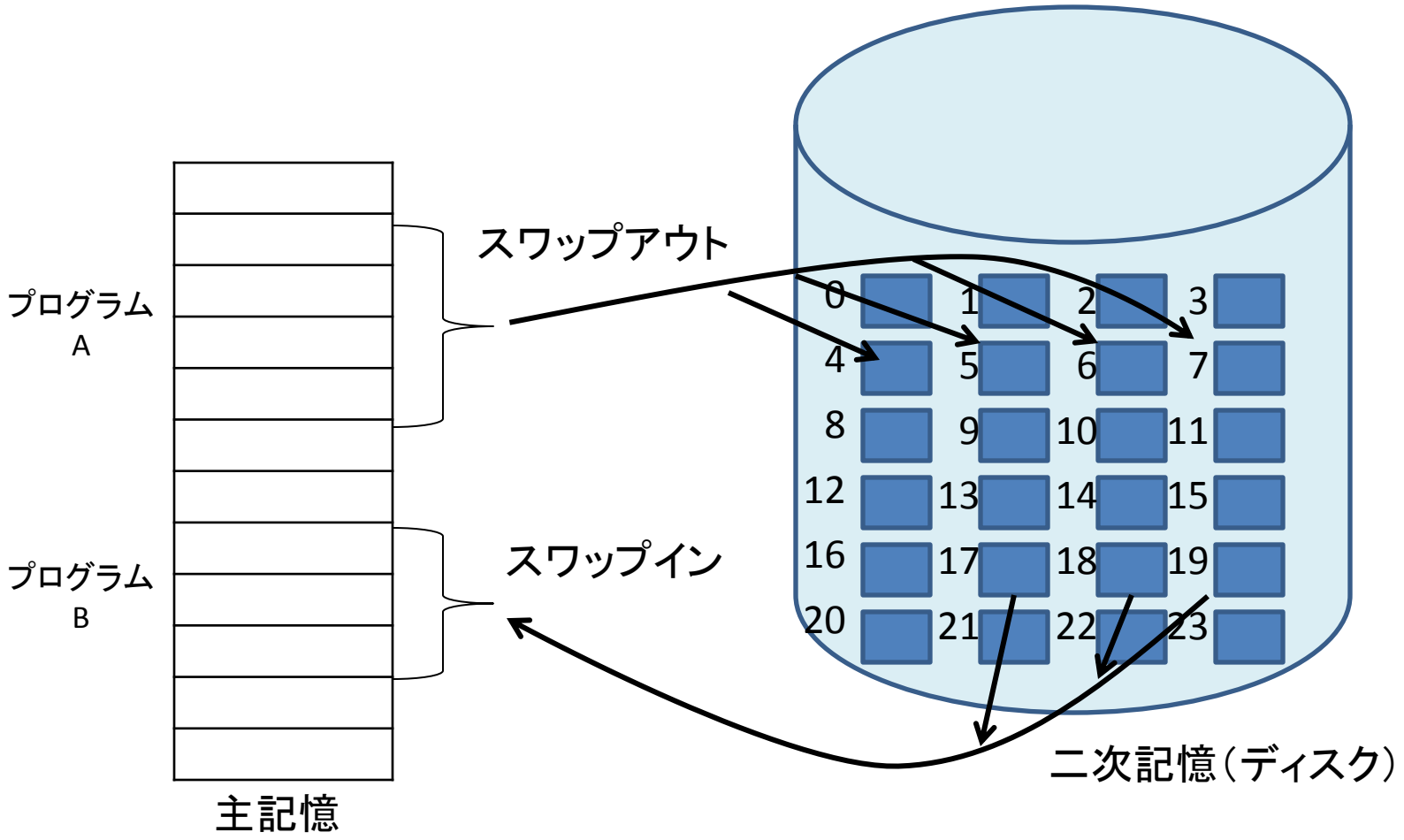


図6．2　連続ディスク領域に対するページ記憶のスワッピング

ページがメモリにないときの処理

- ページ表エントリ内の有効／無効ビット（Valid/Invali bit）で判断
  - 有効／無効ビット：1ビットで，そのページがメモリにあるかないを示す．
  - 有効／無効ビットの値：
    - ✓ メモリにあれば，V(Valid)，無ければ，i(Invalid)と表記する．
    - ✓ 実際は，0，1なのだが，どちらが0か，1か，混同して分かりにくいため．．

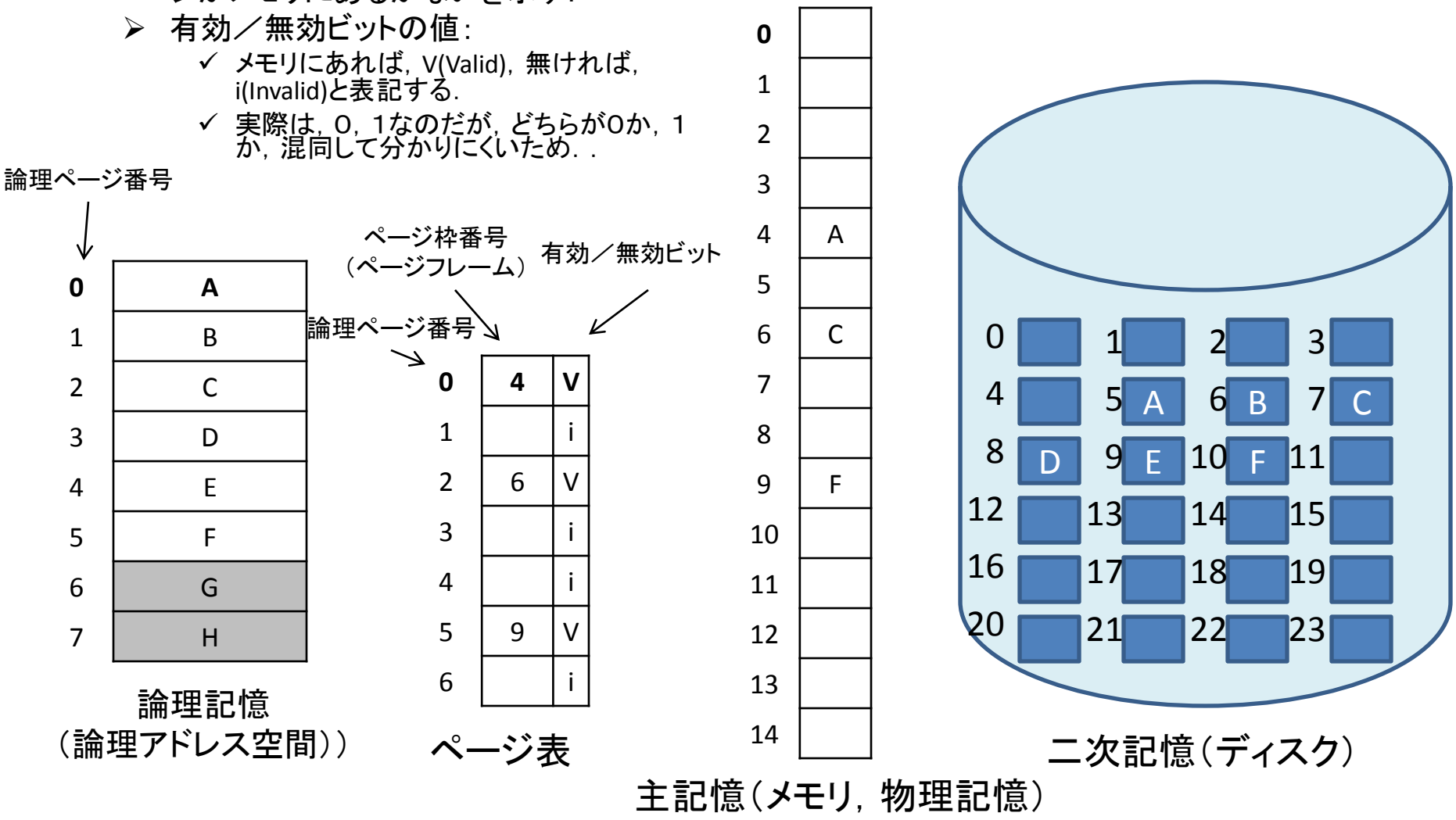


図6．3　ページがメモリ上にないときのページ表の値

ページフォールト処理

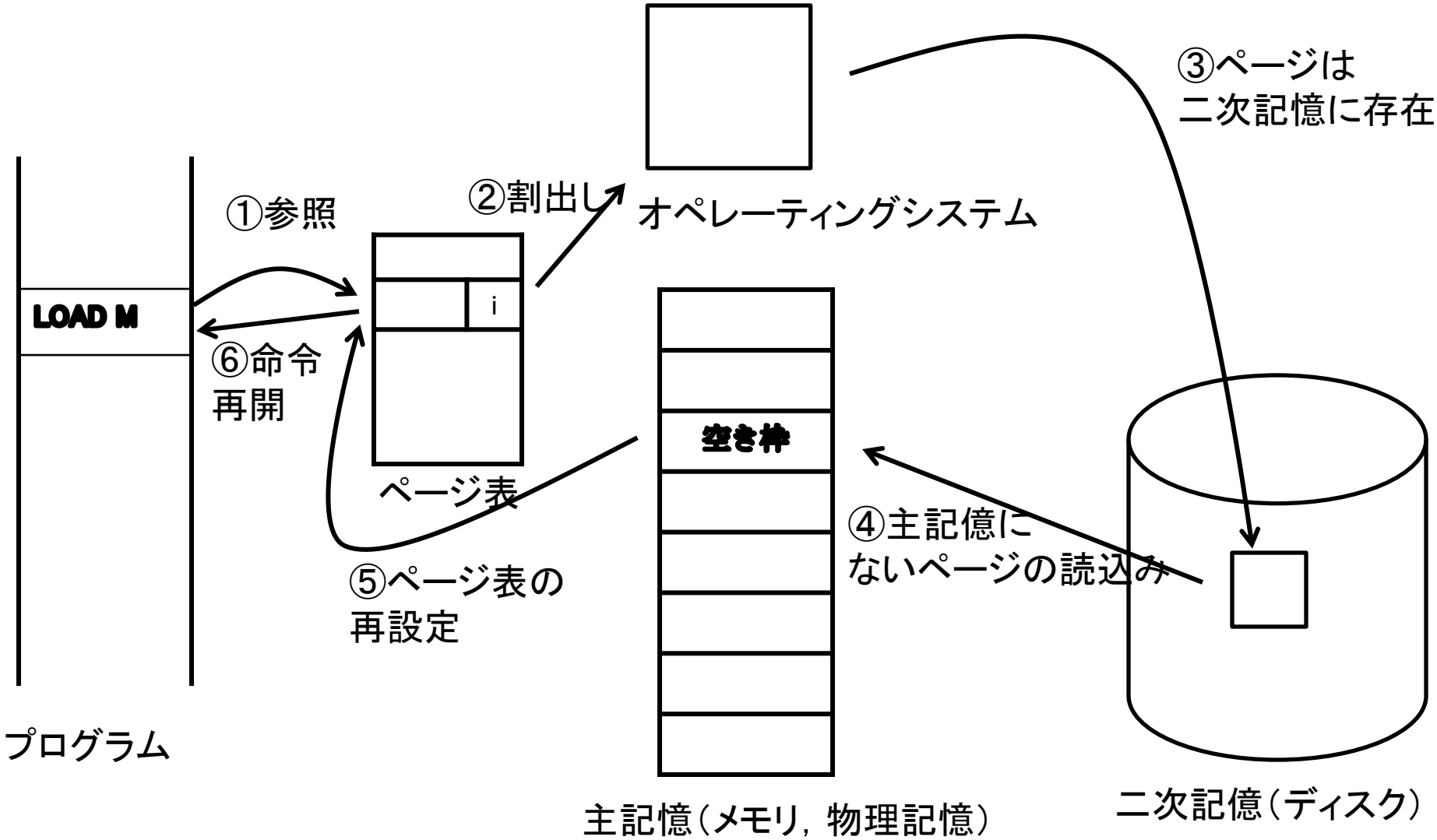


図6．4　ページフォールト処理の手順

ページ置き換え（Page Replacement）

- ページが必要となった.

- ページをメモリにロードする必要あり.

- メモリに空きページフレーム（page frame，ページ枠）がある場合：そのどれかの空きページフレームに入れる．ない場合は，下記.

- 空きページフレームがない場合，メモリ上のどれかのページを追い出して，空きページフレームを作る.

ページフォールトの処理

1. 要求されたページの二次記憶（ディスク）内の格納場所を見つける.
2. メモリ内に空きページを見つける.
   a. 空きページフレームがあれば，それを用いる.
   b. なければ，ページ置き換え（page replacement）アルゴリズムを用いて，犠牲（victim）とするページ）（犠牲ページ，victim page）フレームを見つける.
   c. 犠牲ページを二次記憶に書き出し，その情報を反映するようにページ表の内容を変更する.
3. 要求されたページを（新しい）空きページフレームに読み込み，ページ表の内容を変更する.
4. ユーザプロセスの実行を開始する.

空きページフレームがない場合の処理

●メモリ ←→ 二次記憶：2回のページ転送が必要
- ➢1回：犠牲ページ → 二次記憶へ
- ➢1回：要求されたページ → メモリへ

●犠牲ページ → 二次記憶 の転送
- ●もともと二次記憶にあるページとメモリにあるページが一致していれば，二次記憶に書き出す必要なし．
  - → ページ表のダーティビット（dirty bit）（修正ビット（modified bit），とも言う，）で判断．

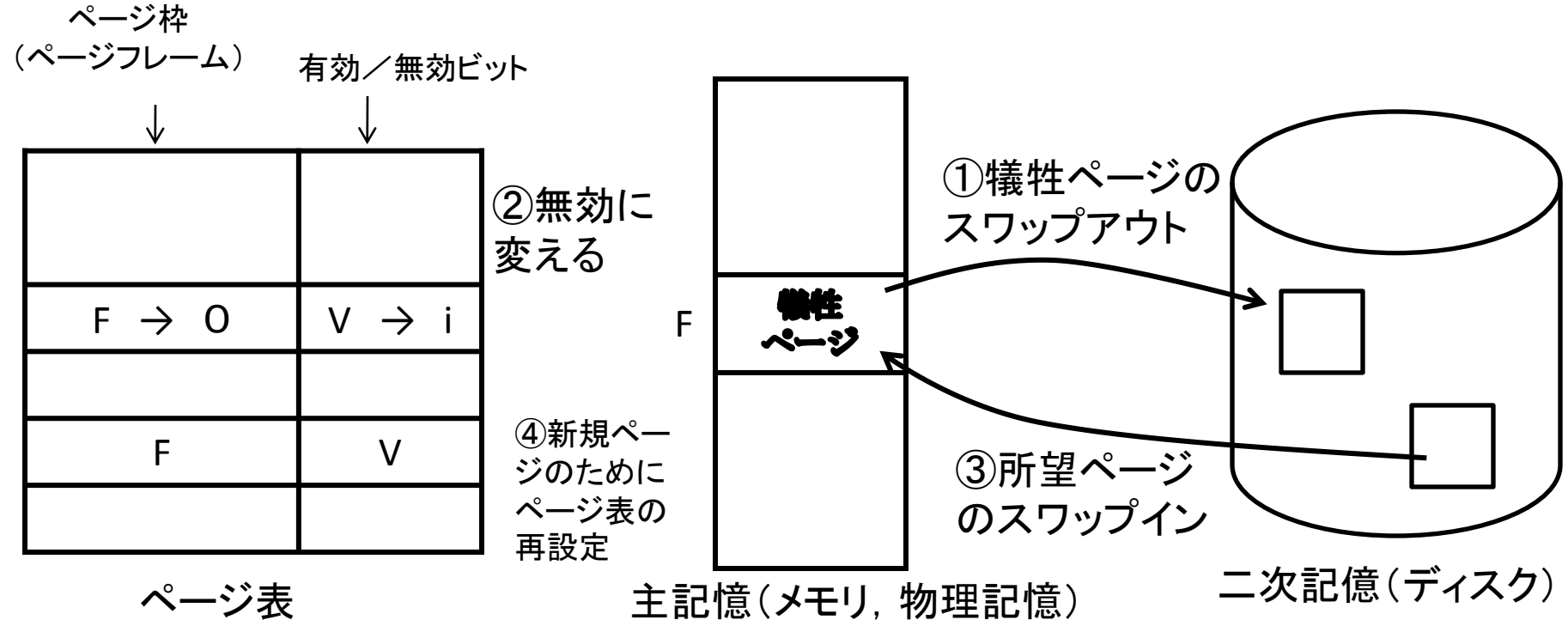


図6．6 ページ置き換え

ページ表のエントリ内の情報

- ページ表エントリとは？
  - 1つのページに関するページ表内の項目
- ページ表エントリの情報
  - 有効／無効ビット：そのページがメモリにあるかないかを示す．1ビット
  - ページフレーム番号：メモリにそのページがある場合，そのページが格納されているページフレーム番号
  - 参照ビット(Reference bit)：そのページにアクセスがあるとオンになるビット．ハードウェアが行う．通常，1ビット
  - 修正ビット（Modified bit,汚れビット，更新ビットとも言う）：そのページに書き込みアクセスがあると，オンになるビット．ハードウェアが行う．通常，1ビット．
  - 保護ビット：次の3種類のアクセスを区別．
    - ✓ リードオンリ，読み書き可能，実行可能
    - ✓ 上記の3種類を区別するため，通常，2ビット

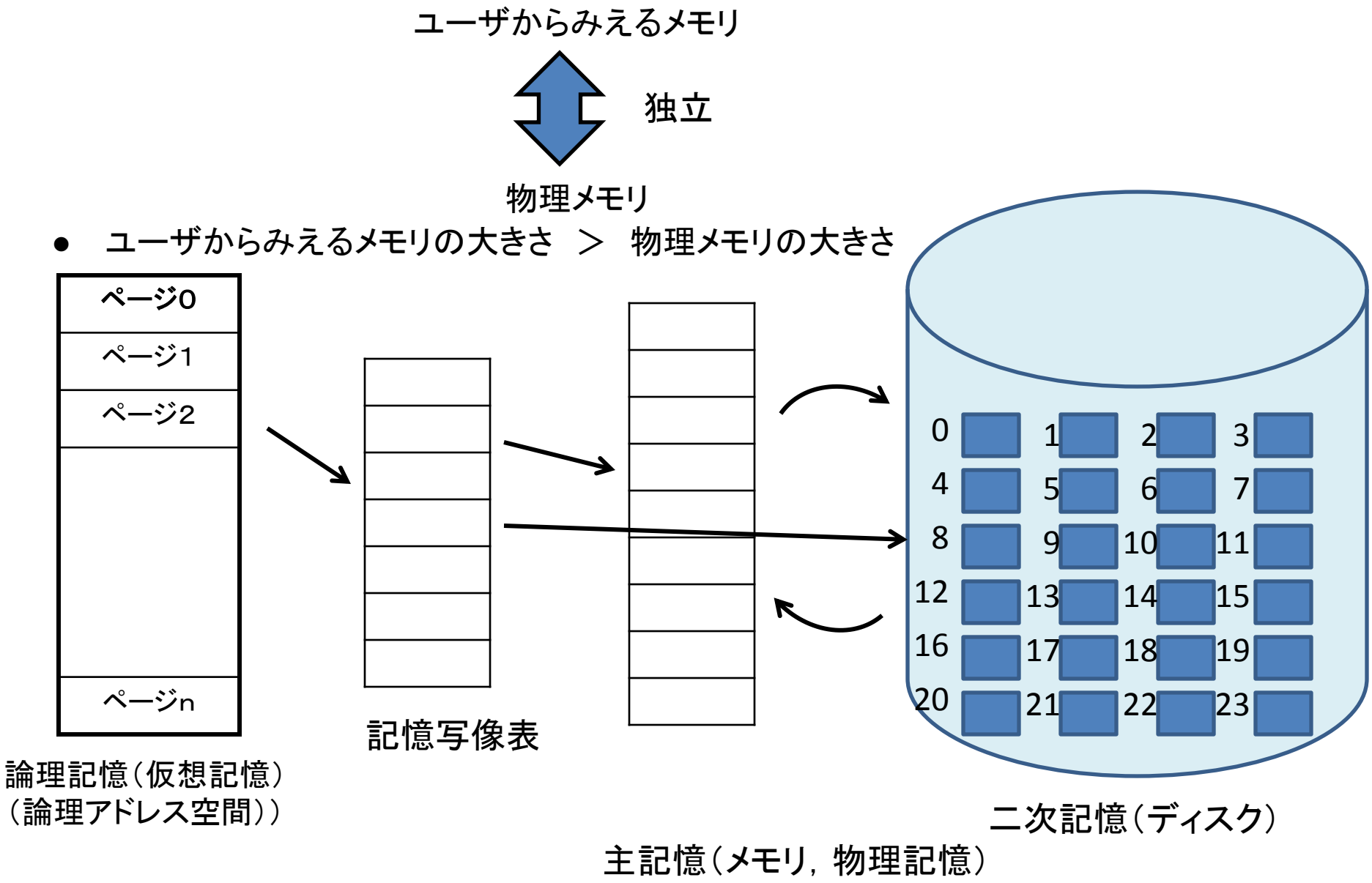
ページ表

ページ表エントリ →

ページフレーム番号	有効／無効ビット	参照ビット	修正ビット	保護ビット

図　ページ表エントリ

仮想記憶の概念



- ユーザからみえるメモリの大きさ ＞ 物理メモリの大きさ

図6．7　物理記憶より大きくできる仮想記憶

ページ置き換えアルゴリズム
（Page Replacement Algorithm）

- 犠牲ページをどうやって選ぶか？

- アルゴリズム

1)FIFO(First In First Out，先入れ先だし)

2)最適(Optimal)置き換え（OPT）

3)LRU(Least Recently Used)

FIFO(First In First Out)アルゴリズム

- メモリにロードされた順に追い出す.
- 例:図６．９　ページフォールト数＝１５回

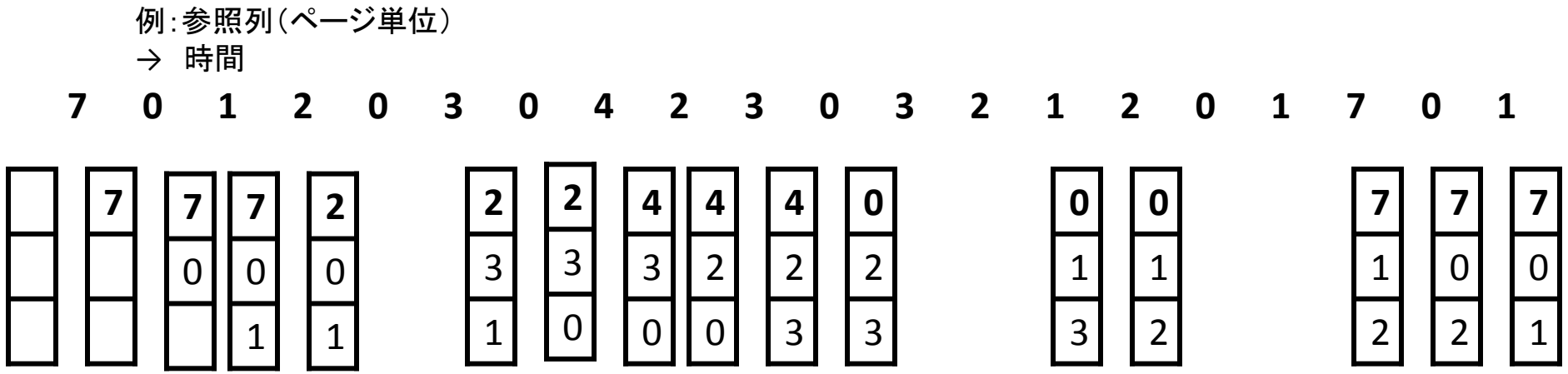


図６．９　FIFOページ置き換え

Beladyの異常振る舞い(Belady's anomaly)

- 直感！

ページフレーム数 → 大
↓
ページフォールト数 → 小

- 事実

ページフレーム数 → 大
↓
ページフォールト数 → 大

になることがある！

例：参照列（ページ単位）
→ 時間

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

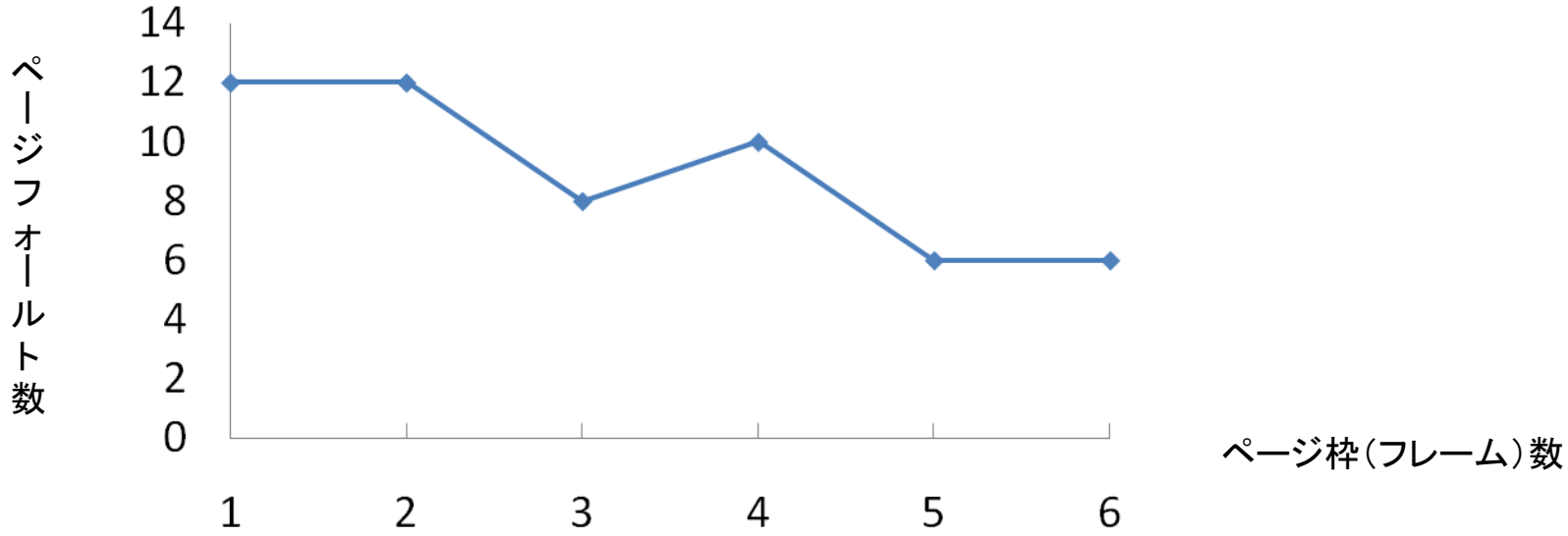


図6．10　参照列に対するFIFOページ置き換えのページフォールト曲線

OPT(最適置き換え)アルゴリズム

- 将来最も長い間使用されないページを置き換える.
- ページフォールト率が最小（理論的）
- 例：図6．11　ページフォールト数＝9回（最小）

例：参照列（ページ単位）
→　時間

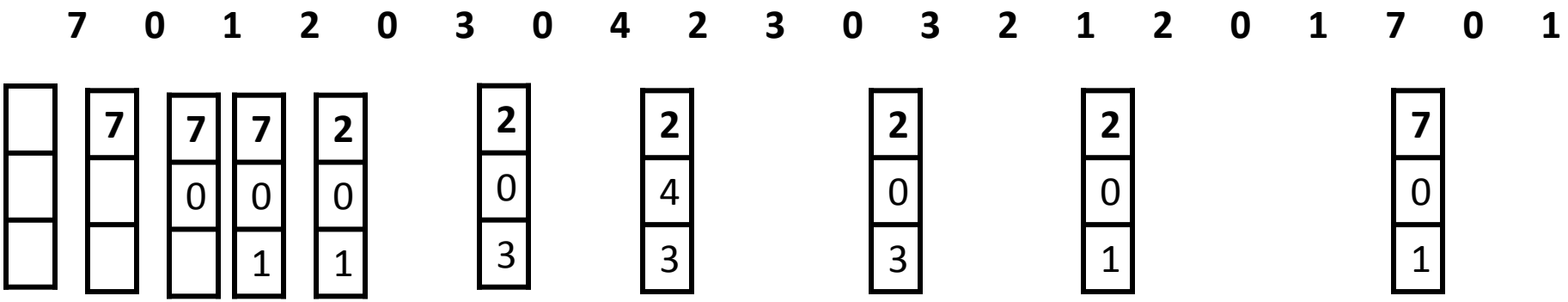


図6．11　最適ページ置き換え

●致命的
　➢将来の情報を必要とする！

LRU(Least　Recently　Used)アルゴリズム

- OPTの近似
- OPT：将来最も使用されないページ
- LRU：最も最近使用されないページ
- 例：図６．１２　ページフォールト数＝１２回

例：参照列（ページ単位）
→　時間

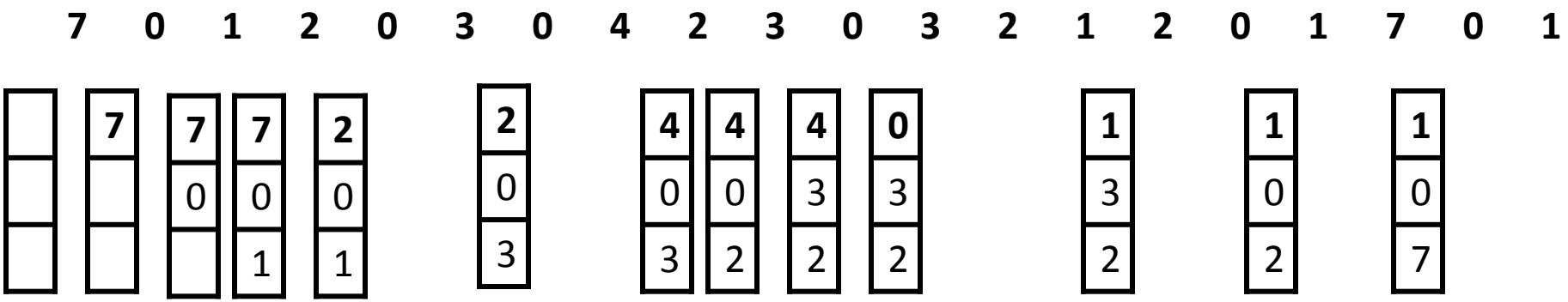


図６．１２　最長未使用（LRU）ページ置き換え

LRUの実現方法

1）カウンタを用いる方法
2）スタックを用いる方法

1）カウンタを用いる方法
- ページ参照ごとに時刻（カウンタ）を記入する.
- ページごとに.

2）スタックを用いる方法
- ページ参照ごとに参照されたページをスタックにおく.
- スタックの上（トップ）：最近参照されたページ
- スタックの底（ボトム）：最も古く参照されたページ

図6．13　最近ページ参照を記録するためのスタックの使用（準備中）

LRUの近似アルゴリズム

- LRUの欠点
  参照ごとにアクションをとる必要あり.
  ↓
  オーバヘッドが大
  ハードウェアのサポートを必要とする.
- 通常のハードウェア
  - 参照ビット(reference bit)を持っている.

  参照ビット：ページ対応にある.
  ページが参照されると（通常，ハードウェアによって）当該ビットがオンになる.
- 付加的参照ビット
  - ハードウェアの参照ビットを用いて，ソフトウェアで実現する.

ハードウェア（参照ビット）	ソフトウェア
1ビット	（例えば）8ビット
1単位時間	8単位時間

Second Chance Replacement（二度目の機会ページ置き換え）

- ポインタ：今度見るページを指している.
- ポインタを動かしながら,
  - ページが参照されていなければ（0），そのページを犠牲ページとして選択する.
  - 参照されていれば（1），クリア（0）して次に進む.

図6．14　二度目の機会ページ置き換え（準備中）

最小使用頻度ページ置き換え（LFU:Least Frequently Used）

- どのページを犠牲ページに選ぶか？
  - 参照回数の最も少ないページを犠牲ページに選ぶ．．
- 発想
  - 参照回数の多いページは，また参照されるだろう（本当？）
  - 参照回数が，将来の使用頻度を反映しているか？
  - （例） 参照頻度にむらがあればダメ：最初に参照頻度が偏るプログラム

最大使用頻度ページ置き換え（MFU:Most Frequently Used）

- どのページを犠牲ページに選ぶか？
  - 参照回数が最大であるページを犠牲ページに選ぶ.
- 発想
  - 参照回数の小さいページは，メモリにロードされたばかりであろうから，これから参照されるに違いない（本当？）
- LFU,MFUとも，実際あまり用いられない.

その他のページ置き換えアルゴリズム

- ページクラスを導入する方法

（参照ビット，修正ビット）
クラス0：（0，0） 参照なし＆変更なし
クラス1：（0，1） （最近）参照なし＆変更あり
クラス0：（1，0） 参照あり＆変更なし
クラス0：（1，1） 参照あり＆変更あり

犠牲ページ
　クラス0から順に選ぶ.

シングルプログラミング環境の場合
ー1つのプログラムしか走らせない場合ー

- 基本
  - プログラムを走らせるとき
    - ✓ まず，全て空きページフレームにしておく.
    - ✓ デマンドページングでページをロード
- 変形（種々あり）

  １．必ず常時空きページフレームを確保しておく.
  ２．ページフォールト発生
  ３．デマンドページを空きページフレームにロード（DMAを用いる）
  ４．犠牲ページを選択（CPUが行う）
  （上記３，４の２つは，並列実行できる）
  ５．ページがロードされると実行再開（CPU）
  ６．犠牲ページをディスクに格納（DMAを用いる）
  （上記５，６の２つは，並列実行できる．）

- マルチプログラミング環境では，どのようにしてプログラムにメモリを割り当てるか？

マルチプログラミング環境の場合
―複数のプログラムを走らせる場合―

- プロセスへのページフレームの割り当てをどうするか？
- 大域割当（Global replacement）
  - ユーザ全体に割当てられてページフレームから選択
    - ✓選択：ページ置き換えアルゴリズム
- 局所割当（Local replacement）
  - プロセス毎にページフレーム（数）を固定
    - ✓当該プロセスに割当てられているページフレームから選択

スラッシング

- 準備中

以上