メモリ管理の後半で、仮想記憶についてです。教科書では第10章です。

オペレーティングシステム
（2024年 第６回）

メモリ管理について（2）

前回の課題について

詳細は「第5回の課題について.pdf」をみてください

「割当て領域のアドレスを要求プログラムに返している」というのは、プログラム実行中に malloc のような処理によってメモリ領域 M を割り当て、そこのアドレス Ma をプログラムに返すことであり、プログラム内では Ma によって M を直接参照・操作します。

ここで、Ma を格納した領域（ポインタ）については、アドレスを格納しているということは実行時に OS からは分からないので、このプロセスの領域を移動すると、（動的再配置がなければ）たとえば 1000番地にあるデータ x がポインタ p によって参照されている場合、詰直しによって x が 550番地に移動されたときに、元々は 1000 というアドレス値を保持していた p の内容を 550 に変更することができず、内容が誤ってしまうことになります。

仮想記憶（Virtual Memory）とは

プログラムが使用するメモリアドレスを実際のメモリ装置のアドレスとは切り離して、論理的なものにする

→ ユーザ（プログラム）から見える実質的なメモリは、実際のメモリ装置や二次記憶装置を用いて実現された「仮想」のもの

- 論理的なアドレス（論理アドレス、仮想アドレス）
  プログラム（データも含めて）に対して与えられるアドレスで、連続した領域を構成する
  - そのプログラム（もしくはプロセス）だけを考えて与えられ、メモリ装置のサイズや他のプログラムなどには影響されない
- 物理的なアドレス（物理アドレス、実アドレス）
  - 実装されているメモリ装置に対して与えられるアドレス

仮想アドレスと実アドレスという言葉についてはこれまでに何回か出てきています。第3回講義資料のp.16などを見直してください。

仮想記憶とは、実際のメモリの制約、サイズやどの部分を使っているか、などを考えず、プログラムはいくつあってもそれぞれは論理的に0番地から始まる連続した空間があって、それぞれ十分なサイズで使用できるようにすることです。

そこで、実際の物理的なメモリアドレス（実アドレス）とプログラムが使用する論理的なアドレス（仮想アドレス）とに分けて考えることにします。

仮想記憶のしくみ

例えば、オーバーレイは主メモリの容量より大きなサイズのプログラムの実行を可能にするが、プログラマがモジュール分割や主メモリでの位置、ロード処理を決定しなければならず、大きな負担になる

- メモリ管理の目標
  - 不連続な物理メモリ領域で連続な仮想メモリ領域を実現
  - メモリ(物理メモリ)より大きな記憶空間の提供
  - オンデマンドの処理
- 仮想記憶

  物理メモリと二次記憶を利用して上記の目標を実現するメモリ管理式
  - プログラムやデータは二次記憶に格納され、必要に応じて物理メモリにもってこられるが、その操作はユーザには意識されない

仮想記憶の大まかな考えは以下です。

前回、メモリの同じ場所を上書きするようモジュール分割を行って、使用するメモリのサイズを制御するオーバーレイについて説明しました。オーバーレイでは、必要なくなればメモリを上書きして使用すればよいということですが、ユーザ(プログラマ)がモジュール分割やどこにいつロードするかなどを考えて決めなければならず、大きな負担です。
負担なく、大きな、連続するメモリ空間を、必要な時に得られるようにすることが目標です。

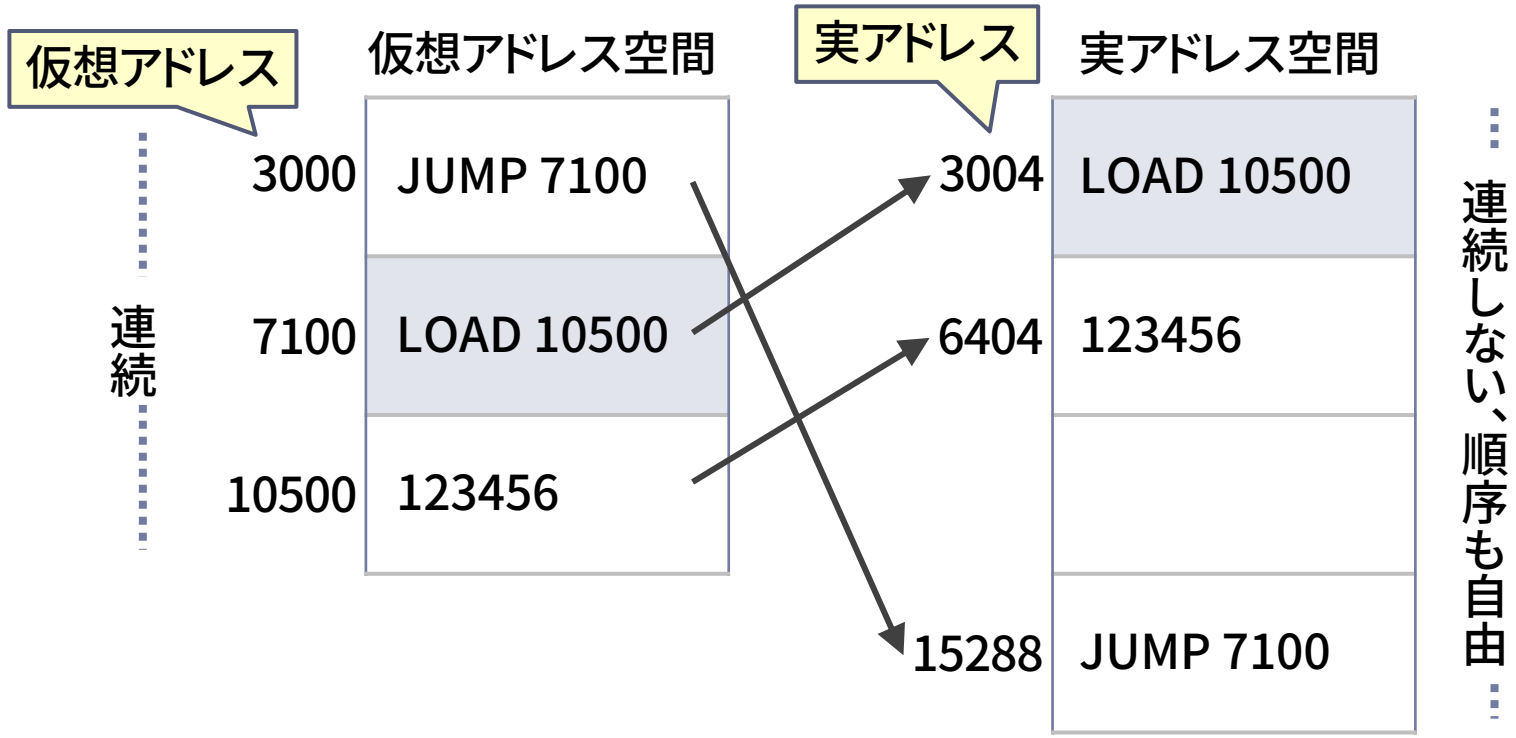
さらに、スワッピングにあるように、二次記憶(ハードディスクなど)をバックアップに使って、実装されているメモリサイズ以上のメモリ空間を実現しようとするものです。

それらの操作はユーザには意識されず、必要な時に必要な部分が実際のメモリ上に持ってこられるようにするものです。

アドレス空間と実際の記憶場所の分離

ページングの導入

- メモリ管理での本質的な制約
  プログラムへのメモリ割り付けは連続的な領域でなければならない
  - この制約が回避できれば、メモリ上に散在する空き領域をまとめて大きな領域とすることができる



まず、連続した領域について考えます。この制約が回避できれば、散在する空き領域を(論理的に)まとまった大きな領域と考えることができます。
図のように、仮想アドレスとしては順番に配置されているプログラムが、実アドレスのメモリ上で散在する場所にあってもよいようにしたいということです。

前回の詰め直しのことを思い出すと、プログラムの部分が連続せずいろいろなところに置かれるとしたら、再配置の問題が出ると思うかもしれません。
以降で、そのことについて説明します。

仮想記憶の基本

プログラムをある単位で分割し、その単位でアドレスの変換とメモリ領域の割り当てを行う

分割のしかたによって2つの方式がある

- セグメンテーション
  プログラム作成者が、プログラムの内部構造に基づいて定義する「セグメント」を単位とする方式
- ページング
  「ページ」と呼ばれる一定のサイズの単位でメモリ領域の割り当てを行う方式
  - ページはプログラムとは無関係にシステムが自動的に分割し、サイズはシステムで固定

プログラムをある単位で分割してその単位ごとにメモリ領域の割り当てと、それに対応してアドレスを仮想から実へ変換するということを考えます。
仮想メモリでのあるブロックが実メモリ上でのあるブロックに対応するというイメージです。

ブロックが可変長であるものと固定長であるものの2つの方式が考えられ、可変なものをセグメンテーション、固定のものをページングと呼んでいます。

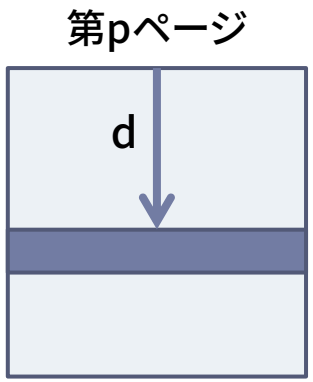
講義では、主にページングについて説明します。
ページングは、「ページ」と呼ばれる一定のサイズの単位でメモリ領域の割り当てを行うもので、ページはプログラムとは無関係にシステムが自動的に分割し、サイズはシステムで固定されています。

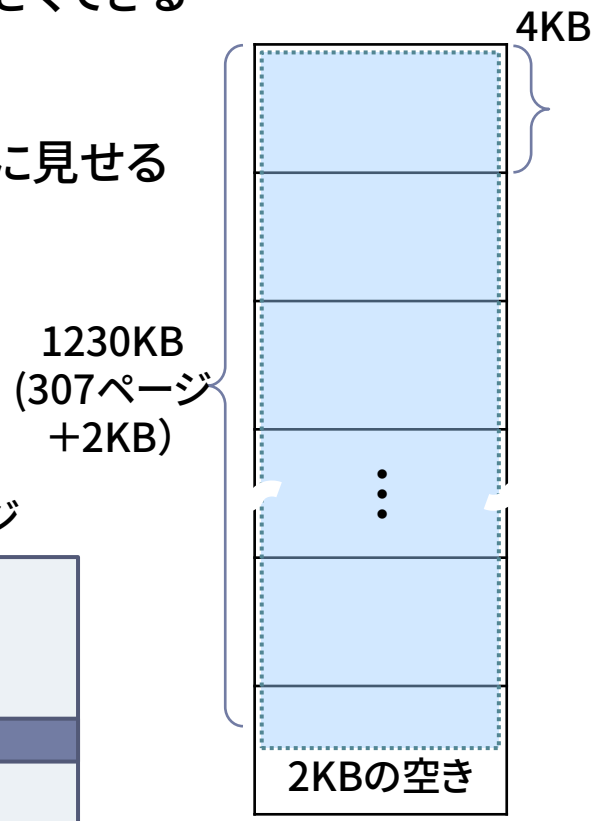
ページング

- メモリのページ化
  - メモリ領域を一定サイズの小さなブロック(ページ)に区切る
    - メモリ内の空き領域をページサイズよりも小さくできる
  - アドレス変換テーブルを用意する
    - 物理的に離れているページを論理的に連続に見せる
  - アドレスを論理化する
    - 論理アドレス(仮想アドレス)を物理アドレス(実アドレス)に変換する
  - アドレスの表現

ページ番号 p	オフセット d

pページの先頭からd変位した位置





メモリ領域を一定サイズの比較的小さなブロック(ページ)に区切ります。
これによって、メモリ内の空き領域をページサイズより小さくできます。ページサイズは4KBが典型的なものですが、1230KBのプログラムサイズを保持するには308ページ必要で、最後のページに2KBの空き領域が生じるだけとなります。

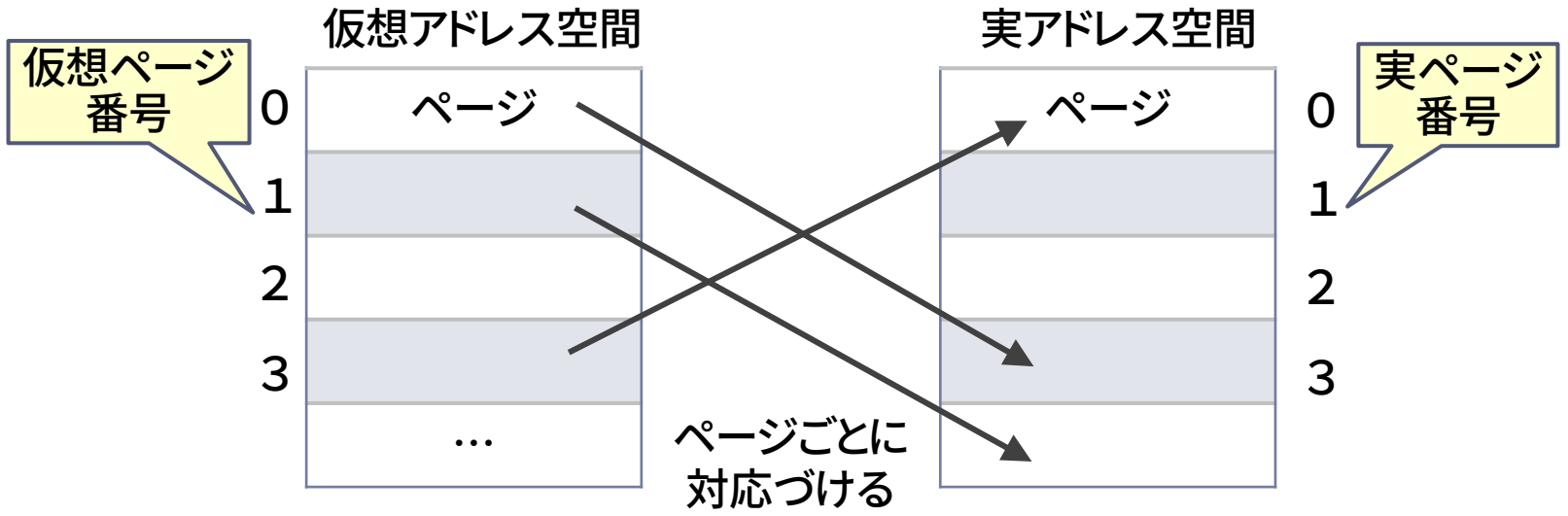
ページが離れたところにあっても、アドレス変換を行うことで連続しているようにみせます。

アドレスは、ページ番号とページ内の先頭からの変位(オフセット)で表現されます。4KBのページサイズとすると、アドレス20000は4ページ(0ページから開始するものとします)の先頭からオフセット3616として、<4, 3616>のように表されます。

なお、以降でも説明では、K(キロ)を2^{10}=1024、M(メガ)を2^{20}=1048576、G(ギガ)を2^{30}= 1073741824を表すものなどとしています。

ページングとアドレス変換

- ページングとアドレス変換
  - ページで構成されるアドレス空間
    - 仮想アドレス空間と実アドレス空間が同一のページ単位で分割
    - 各ページには、仮想ページ番号と実ページ番号が付与
    - ページとプログラムの論理構造とは無関係
      コードやデータがページに跨って配置



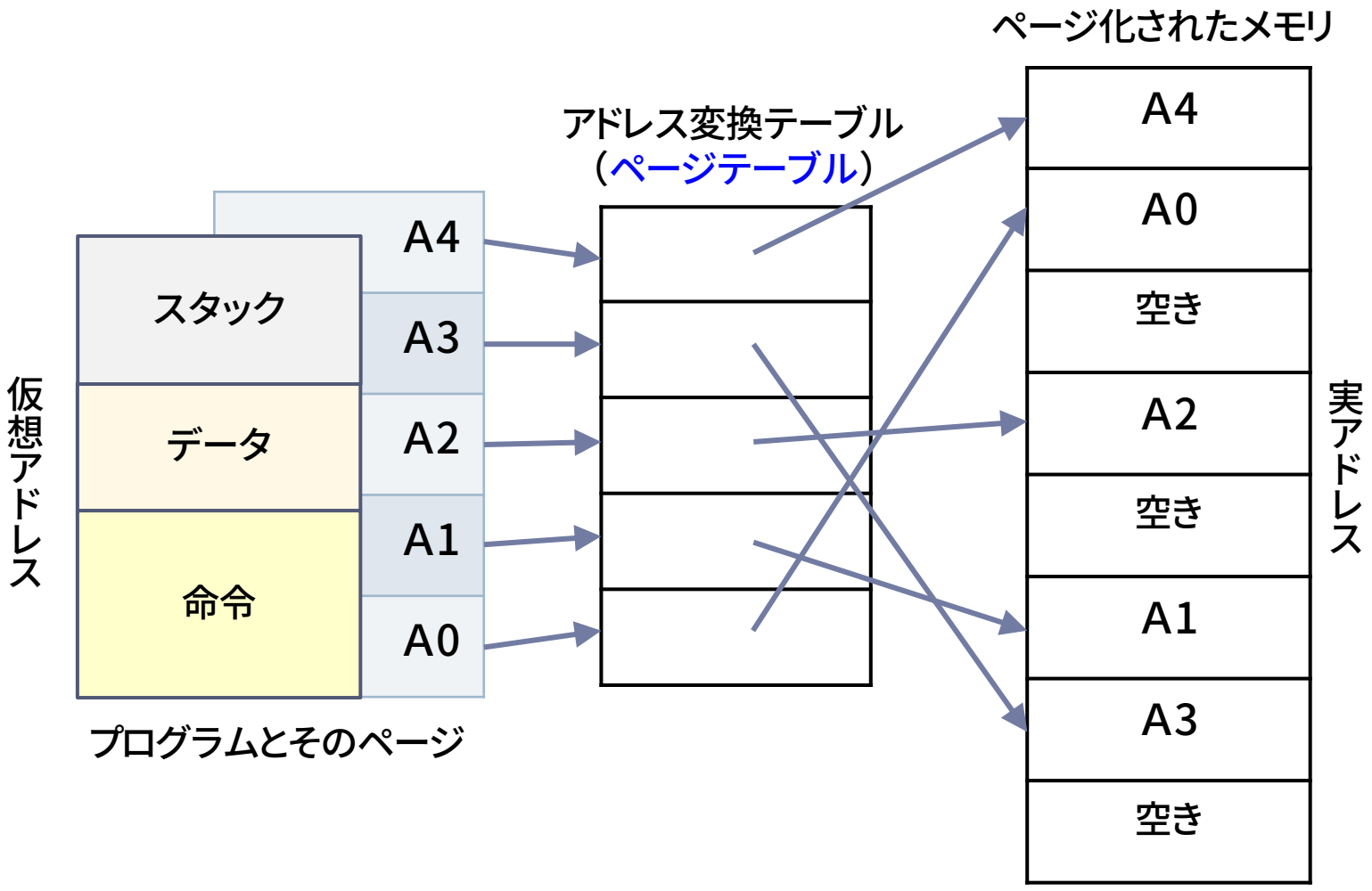
  - 断片化が解消される

仮想アドレスの空間、実アドレスの空間それぞれを同じサイズのページで分割します。（たとえば4KBのサイズのページの連続で構成されているとすることです）
それぞれには、仮想のページ番号、実のページ番号が割り振られます。

ここで、ページとプログラムの論理的な構造は無関係です。ページの境界で命令やデータが区切れている必要はありません。
仮想アドレス空間の0ページが実アドレス空間では3ページに対応するというような関係が付きます。

前のページの1230KBの領域のように、最後のページにページサイズ（4KB）より小さな領域が使用されずに残るだけなので、断片化がほぼ解決できます。

アドレス変換テーブル

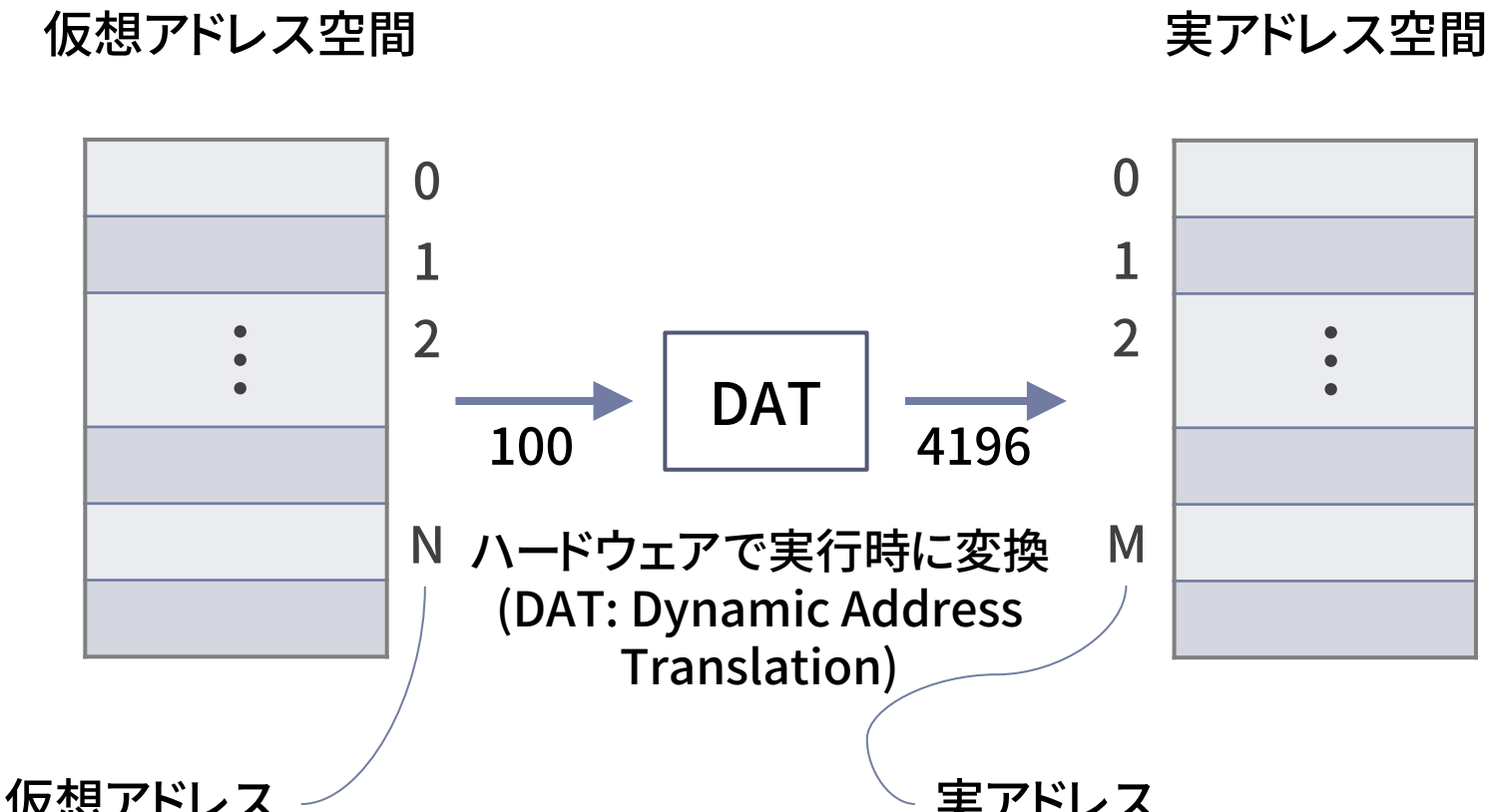


以上を図にするとこのような形で、仮想アドレス上で命令やデータがA0、A1、・・・の内容で順に構成されており、実空間上ではA0の内容のページが上から2番目など、不連続で順序もバラバラな場所にあるようになります。

この対応をとるものが、アドレス変換テーブルです。第3回講義資料のp.17を再度見てください。そこのDATでアドレス変換テーブルを参照します。

このアドレス変換テーブルをページテーブルと呼びます。

ハードウェアによるアドレス変換（第3回）

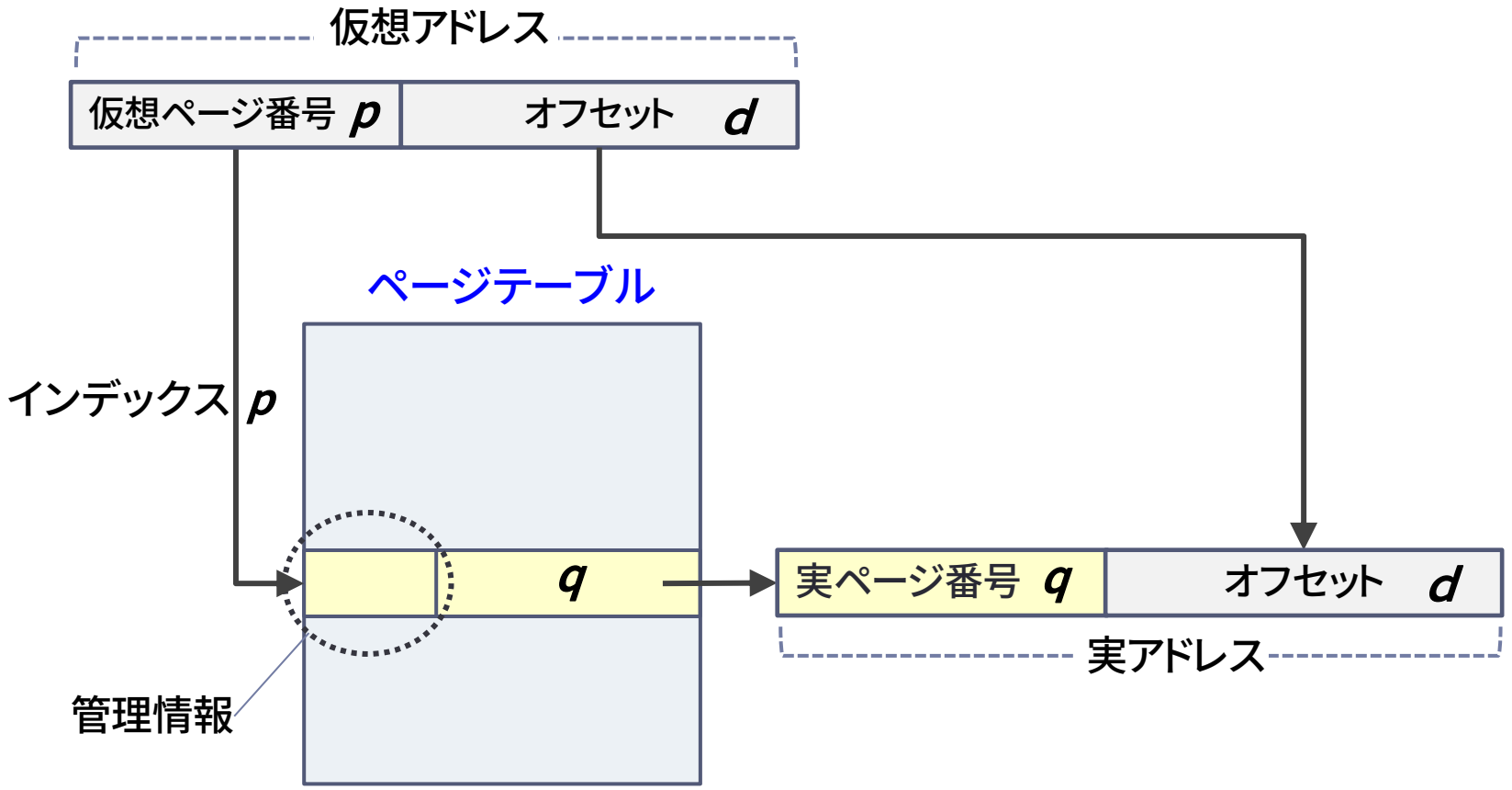


プログラムの実行時に、仮想アドレスから実アドレス（物理アドレス）へのアドレス変換が、ハードウェアで行われます。
アドレス変換を行う機構は、動的アドレス変換機構：DAT（Dynamic Address Translation）と呼ばれ、CPUの中にあるメモリ管理ユニット：MMU（Memory Management Unit）で行われます。
この図では、プログラムの実行時に、プログラムでは100番地にあるものとする変数を参照するのに、CPUが「100」を出力すると、それがDATで「4196」と変換されて実際のメモリの4196番地を指定するようになることを示しています。

このような変換を行うことで、仮想アドレス空間では同じアドレス（複数のプログラムのものなど）が異なる実アドレスに対応できるようになります。

ページテーブルによるアドレス変換

- 仮想メモリから実メモリ空間への対応づけはページテーブルと呼ばれる構造で実現



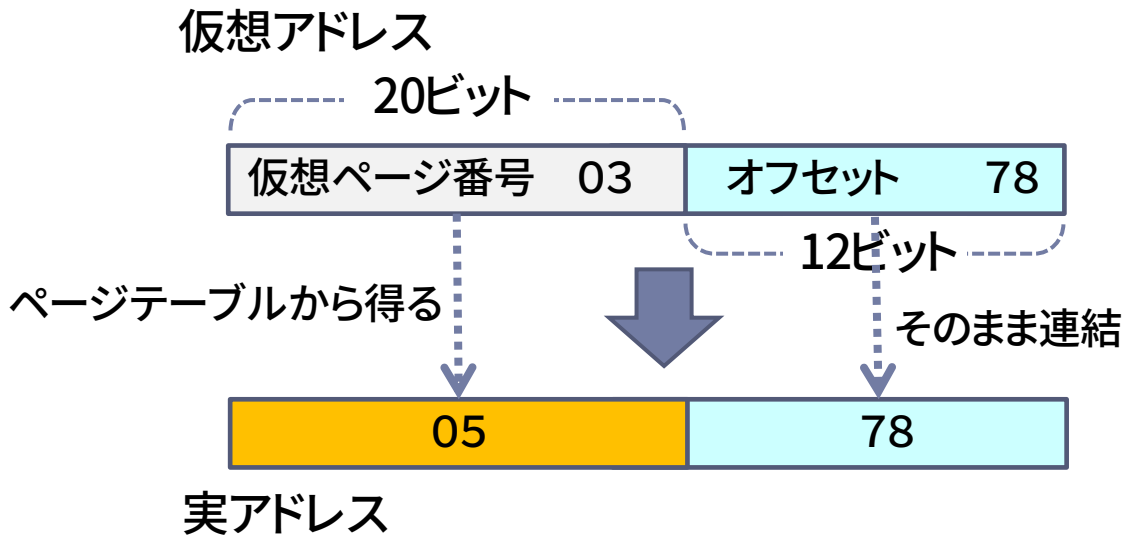
ページテーブル(アドレス変換テーブル)は、このような構成になっています。
ページで分割しているので、仮想アドレスは仮想のページ番号とページ内のオフセットという形式で表現されます。ページサイズが4096バイトとすると、アドレス10000は、2ページの先頭から1808バイト行ったところという具合です。

ページテーブルは、実ページ番号を要素とする配列で、仮想ページ番号pをインデクスとして引いた要素の値が、対応付けられる実ページ番号qとなる(ように設定する)ものです。
実アドレスはqと元々のdで構成されます。
上の例で、変換された実ページ番号が15とすると、15*4096+1808 = 63248が実アドレスになります。

ページテーブルのエントリには変換用のページアドレスだけでなく、管理用のデータも入っています。

基本機構（DATでの変換）

- 仮に、32ビットのアドレス空間でページサイズを4KBとすると、ページ番号は20ビット、オフセットは12ビット



- ページテーブルによる対応関係の管理
  - CPUのメモリへのアクセスは必ずこの機構（DAT）を経由する
  - ページテーブルが参照されて、仮想ページに対する実ページ番号が得られる
  - 得られた実ページ番号にオフセットを連結して実アドレスが得られる

仮に32ビットのアドレス空間であるとして、ページサイズを4KB（4096バイト）とします。すると、オフセットが12ビットで表現され、ページ番号を保持するには20ビット必要です。

仮想ページ番号「03」をインデクスとしてページテーブルを引いて、3番目の要素の値として実ページ番号「05」が得られます。これとオフセットを加え（連結し）て実アドレスを得ます。

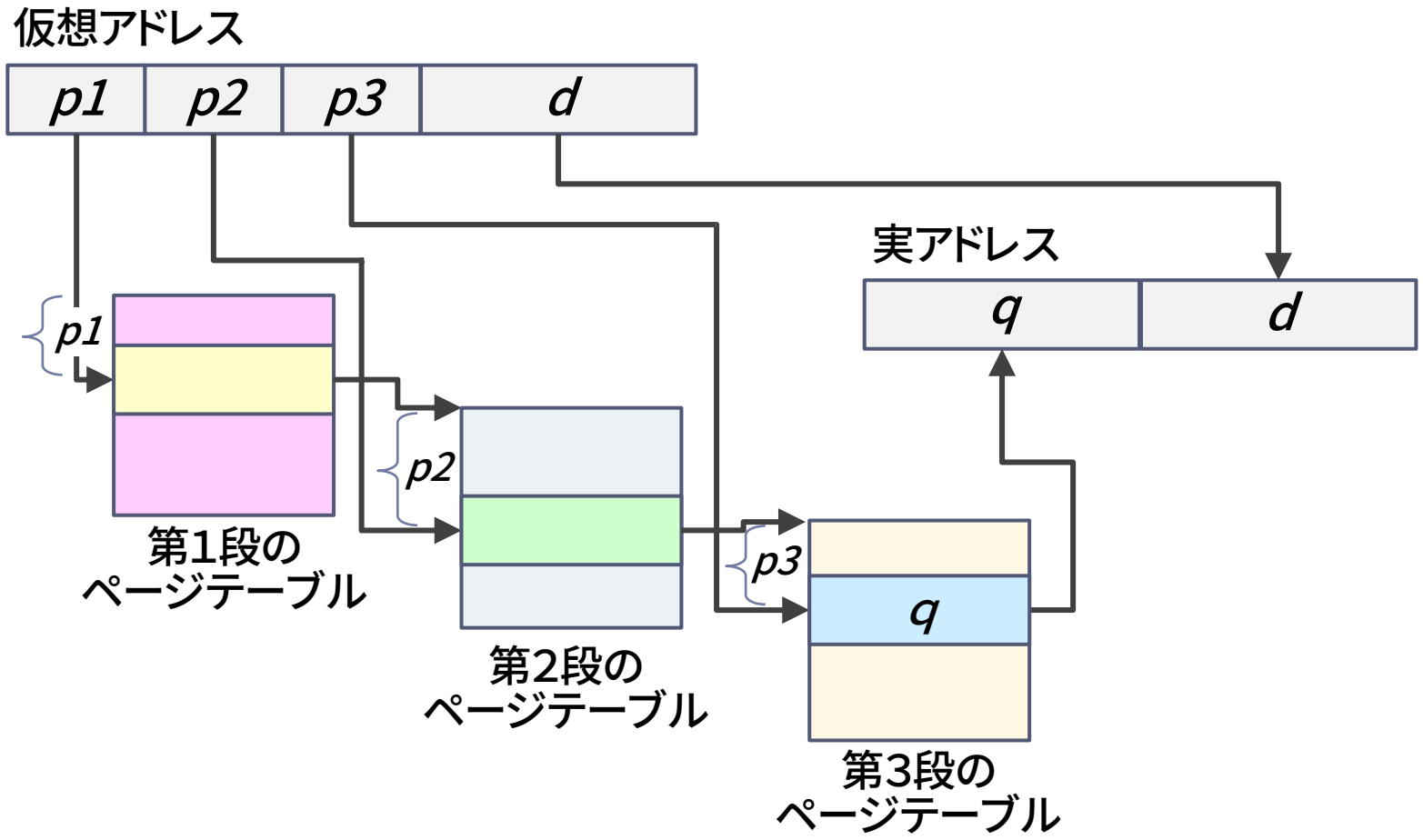
多段のページテーブル

- 前のスライドでは 32ビットのアドレス空間としたが、4KBサイズのページとすると、ページテーブルの大きさは20ビット分の個数×4バイト（＝4MB）程度となる
  （ページ番号が20ビットで、管理情報が何ビットか付加されるので、ページテーブルの1エントリのサイズは4バイト）
- ページテーブルはプロセスに対して1つあるので、100プロセスあれば、400MBがページテーブルに費やされることになり、無駄が大きい
  → ページテーブルを多段にして対処する
- 必要な部分だけページテーブルをもてばよい
  必要なページテーブルだけメモリに置いておくことができる
  - ページテーブルを共用できるので、メモリが節約できる

20ビットのページ番号では、ページテーブルは2^{20}個の要素があり、4MB程度の大きさです。実はページテーブルはプロセスごとに別々にもつので、4MBが1個だけであればそれほどでもないですが、100プロセスでは400MBとなりすべてを持つことは現実的ではありません。
そこで、ページテーブルを何段階かにわけて持つことにします。

アドレス変換の例(1)

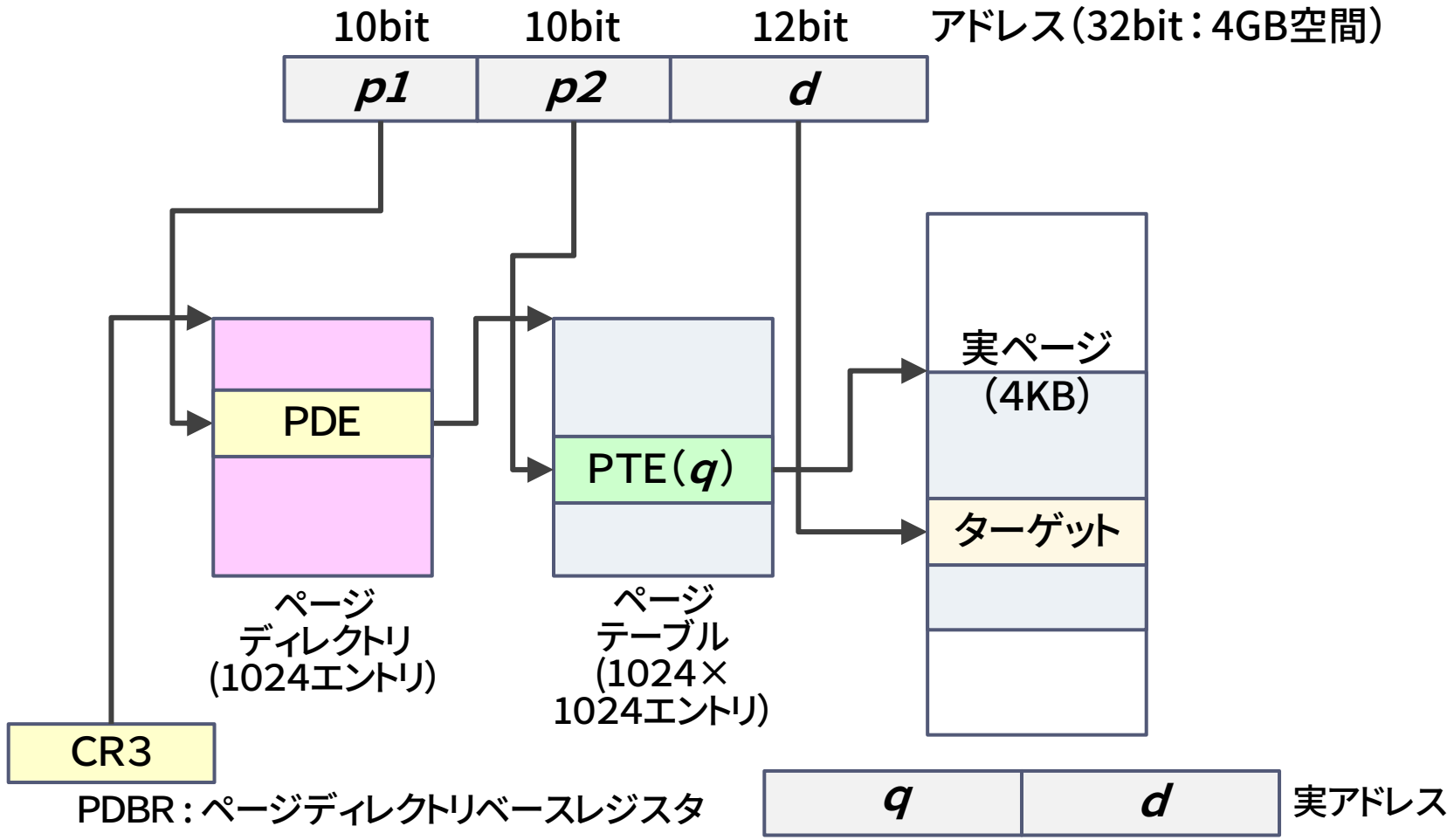
- 3段のページテーブルによるアドレス変換



3段でページテーブルを持つ場合、この図のようになります。変換テーブルの要素の値が次のテーブルの先頭となり、最終のものが変換された実ページ番号になるようにします。

アドレス変換の例(2)　ハードウェアでのサポート

- IA-32の2段のアドレス変換



ページをプログラムでたどるのは処理時間がかかるので、CPUはこれをハードウェアでサポートしています。

IA-32の場合、ハードウェアとしては2段階のテーブルをサポートするようになっています。
PDBR(CR3)という特別なレジスタが一段目のテーブルの開始番地を指し、p1、p2と順にインデクスを引いて実ページ番号を得、それから実アドレスを得るところを自動的に行います。

OSは、CR3の値を設定し、各テーブルの内容を書き込むことを行います。

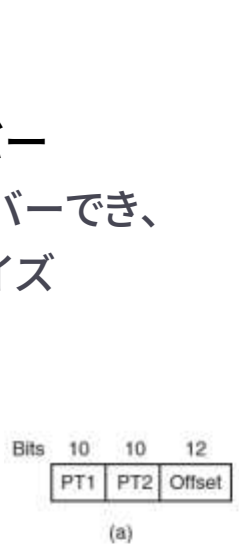
OSで前ページの3段の構成の場合にこのハードウェアを利用するには、2段目のテーブルのサイズを 0 として無視し、1段目と3段目を使用することになります。

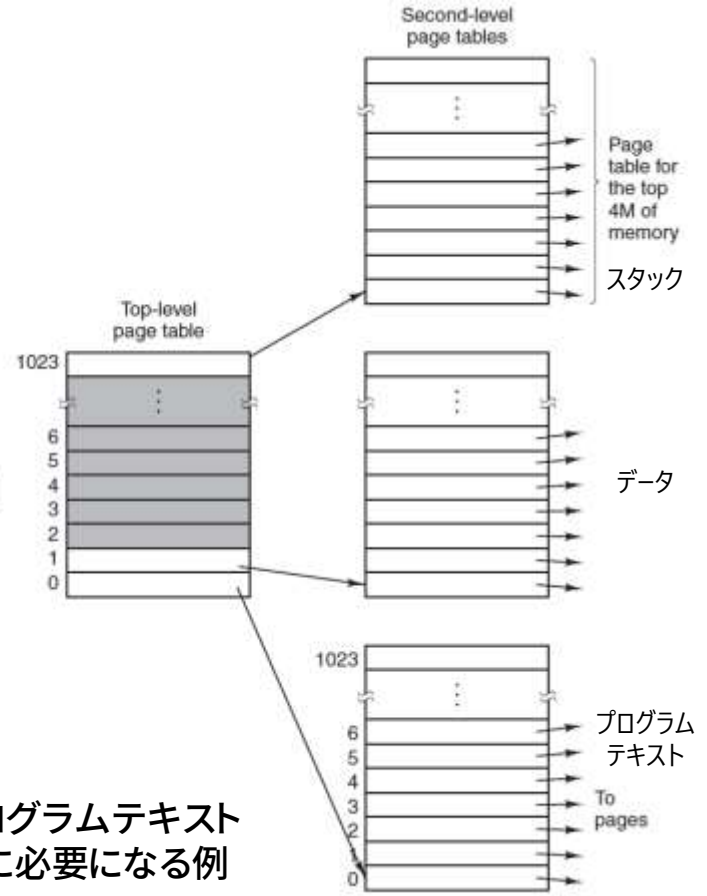
多段のページテーブルでのメモリ削減量

- 1段だけのページテーブルでは、ページテーブルの大きさは4MB
- 一方、2段のページテーブルで、p.14のように1段目、2段目をそれぞれ10ビットのサイズで持つことにすると、1段目は2^{10}個のエントリを持つテーブルが1つで4KB（1エントリのサイズは4Bとする）
- 2段目も2^{10}個のエントリを持つ4KBのテーブルがいくつか必要
  テーブル1個で4KBページが1024個もてるので、4MBのメモリ空間をカバー
  プログラムのサイズが100MBの場合、2段のページテーブは25個あればカバーでき、
  1段目が4KBで2段目が4KB×25＝100KBから、その和104KBが必要なサイズ
  （4MBに対して約2.5%）
- アドレスを変換する時点で必要な2段目のテーブルがメモリ内にあればよいので、1個でも間に合う
  → 最小のテーブルサイズは4KB+4KB=8KBになり、非常に大きな削減
- 2段目のページテーブルがメモリに存在しないときに1段目のページテーブルでページフォールトを発生させる
  → 2回のページフォールトが発生するので、ページアクセスでのオーバーヘッド増加

2段のページテーブルにした場合のページテーブルに要するメモリ量は左のように考えられます。

（2段目のテーブルで不要なものをもたなくて済むから減少するのであって、2段目のテーブルをすべて持つのであれば、減少しません）





右図は参考書「モダンオペレーティングシステム」（原著4版）からで、スタック、データ、プログラムテキスト領域がそれぞれ4MBの領域を必要とするもので、2段目のページテーブルは3つだけが実際に必要になる例

デマンドページング

- デマンドページング
  - 実ページをいつ割り当てるか … 実際にその領域をアクセスしたときに初めて実ページを割り当てる（メモリの利用効率が良い）
    - ロードやシステムコールによってメモリ割り当ての要求があったとき、仮想空間上では、その大きさの領域を確保する
    - 実ページが割り当てられるのは、領域に対する読み書きがあったとき
  - デマンドページングのための機構
    - ページテーブルのエントリに実ページが「未割当て」であることを示すビットを付加
    - 実ページ未割当てで、アドレス変換不能時に例外（ページフォールト）を発生させる
  - オペレーティングシステムにおけるデマンドページングの処理
    1. ユーザプロセスでページフォールト発生
    2. カーネルに遷移し、実ページを割当てる
    3. 例外（ページフォールト）から復帰し、ユーザプロセスを再開

実ページは、実際にそのページがアクセスされるときに存在すればよい。（実行する命令の部分のところ、アクセスするデータがあるところ、だけ）
これによって、実際のメモリ容量より大きなプログラムや、巨大な領域のデータに対応できます。

オンデマンド（要求時）に実ページを割り当てる。
そのため、ページテーブルのエントリに対応する実ページがまだ割り当てられていないことを示す情報を付加します。ページテーブルの説明（p.11）であった「管理情報」を思い出してください。
実ページが未割当ということは、アドレス変換ができないということなので、例外（ページフォールト）を発生させます。
例外処理で、カーネルで実ページを割り当てます。
例外から復帰すると、実ページがあったものとして今度はアドレス変換ができるわけです。

ページングと仮想記憶

メモリ容量を超える場合への対処

- スワッピングを用いる

  （メモリ領域が不足すると実行中のプロセスの全領域を二次記憶装置にスワップアウトして強制的に空き領域を作り、そこに新しいプロセスをロードする）

- プロセスではなく、ページに対してスワッピングを実施する

  メモリ領域が不足すると実行中のページを二次記憶装置に書き出して強制的に空き領域を作り、そこに新しいページをロードする

1つのプログラムで実装されているメモリの容量を上回る場合、ページ単位にスワッピングを行って、強制的に空きのページを作ってそこにこれから実行するページをロードすればよいことになります。

ページングと二次記憶装置の組合せ

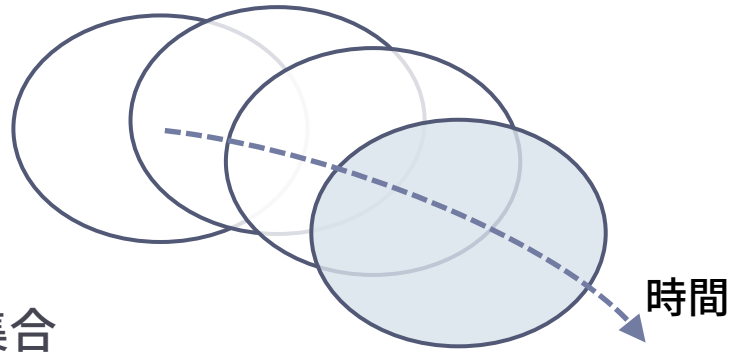
- ページングと二次記憶装置を組合せた仮想記憶
  主メモリの容量を超えてプログラムを実行する
  - ページを二次記憶に置くことで、二次記憶装置の容量まで利用できる
- ページインとページアウト
  - 実ページで未使用のものがなくなる
    → 使用中のページから適当なものを候補を選び、二次記憶装置に書き込み「空き」を作る（ページアウト）
    - ページアウトするページの選出では、この先最も長い間使われないものを選ぶことが得策 →「最も長い間参照されていないページ」
      LRU（Least Recently Used）アルゴリズム
  - ページアウトされた仮想ページが参照されると、ページフォールトが発生し、実ページに読み込まれる（ページイン）

ページに対してスワッピングを行うことで、二次記憶との転送に要するオーバーヘッドを抑えて、実装されているメモリの容量を超えて（二次記憶の容量まで）メモリを使用することができます。

ページに対するスワッピングをページイン、ページアウトと呼びます。どのページをページアウトするかが問題となりますが、LRUと呼ばれるアルゴリズムが多く用いられます。

参照の局所性

- メモリ参照動作と性能
  ページフォールトのたびにページアウト/ページインを繰り返すとプロセスの進行を妨げる
  → 近い将来参照されることがない実ページをページアウトできれば有効
  → プロセスに割り付けられた実ページの活用度を評価する
  - メモリスケジューリング
    … どのページを実メモリに保持し、どのページを二次記憶装置に置くかの選択
  - 参照の局所性
    … プログラム実行の過程でアクセスされるページ群は通常、ある時間の範囲で固まっており、それが時間とともに動いて行く
  - ワーキングセット
    … 一定時間内に参照されるページの集合



ワーキングセットの移動

ページイン・ページアウトを繰り返すとプロセスの進行が妨げられますので、今後参照されないページをページアウトできれば効率的だと考えられます。

通常、プログラムの実行の過程でアクセスされるコードやデータには局所性があり、近い時間に同じ範囲内で発生する確率が高いです。
局所性を活用してどのページを実メモリに保持し、どのページを二次記憶装置に置くかの選択を行うことがメモリスケジューリングです。

ページ置換えアルゴリズム

- LRU（Least Recently Used）… 近頃最も使われなかったページを追い出す
  近い将来に参照される確度が低いと判断する
  - 「参照」ビットを利用し、一定時間ごとに参照ビットを検査して決定する
    ページがアクセスされたら「セット」、検査されたら「リセット」される
    … 近似的な方式（参照した時刻を記録し、それに基づいてページをソートすることは実装上困難）
  - プロセス個々のページ参照特性は考慮しないものが多い
- FIFO … 最も古いページを追い出す
  - 単純なキューで実現できる
  - 頻繁に使用するページでもページアウトされる
  - 実ページ数が増えるとページフォールトが増加する場合がある（Beladyの異常）
- OPT（Optimal）… 将来、最も長い期間使用されないページを追い出す
  - 最適なアルゴリズム：ページフォールト率が最低
  - 将来のページ参照が分かる必要あり

メモリスケジューリングで追い出すページを選択するアルゴリズムをページ置き換えアルゴリズムと呼び、FIFO、OPT、LRUなどがあり、LRU（近似的なもの）が多く使われています。

LRUという用語は前々回（プロセスのスケジューリングで）出てきています。

ページフォールト率は、ページ参照回数に対するページフォールト発生回数の割合で、ページフォールト率が上昇するとページの置き換えに要する時間が多くなり、メモリアクセスに要する時間も増大します。

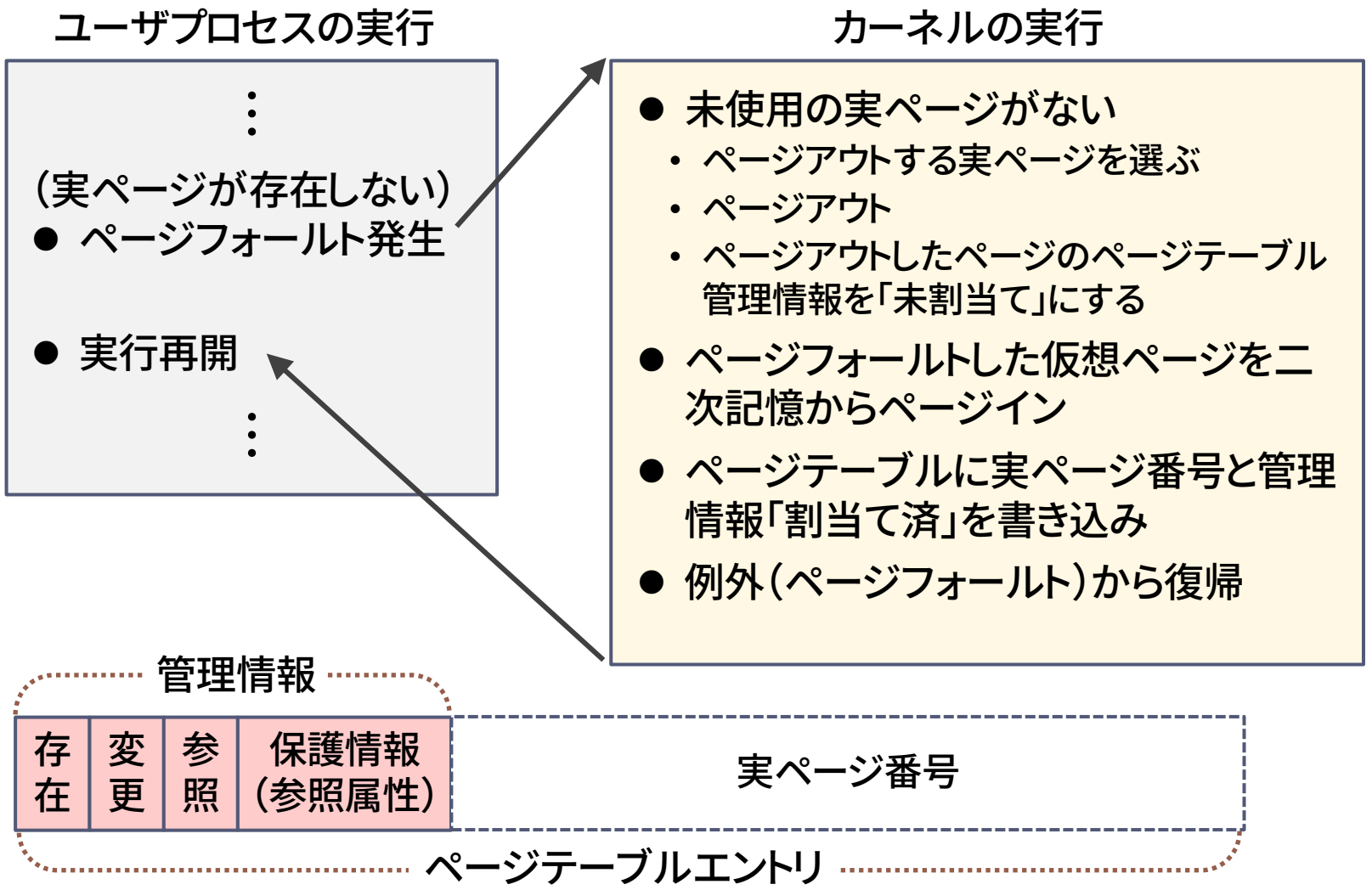
- ワーキングセット法
  … プロセスに割り付けられている実ページの中でワーキングセットの外に出たページを追い出す
  - プロセスに与えられたCPUの時間軸を基本とする
- スラッシング
  ページの置き換えが頻繁に発生し、処理が進まない状態

参照の局所性、ワーキングセット、LRUアルゴリズムについては、ページングやバッファリング、TLB、キャッシュなどに対する基本的な考え方とモデルを提供している

ワーキングセット法は、ワーキングセットに実ページ割り付けを保障することでページフォールトの発生を最小化できるという仮定に基づいています。

スラッシングは、アクセスの際にページの置き換えが頻繁に発生し処理が進まない状態で、プロセスの処理に時間がかかってしまうことです。

仮想記憶の処理の流れの例



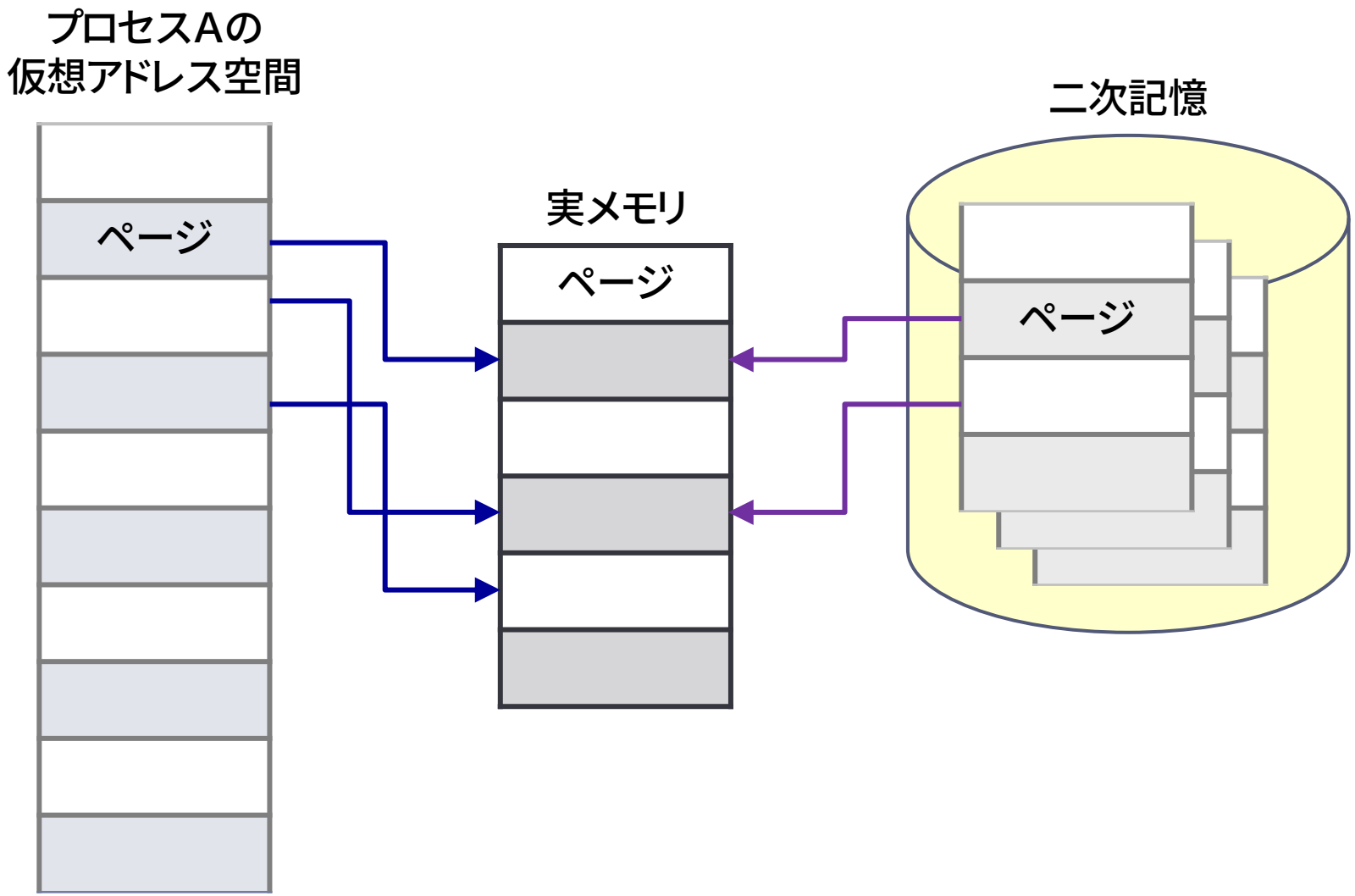
仮想記憶処理の流れをまとめます。

ユーザプロセスでメモリをアクセスすると、DATによってアドレス変換が行われます。このとき、実ページが存在しないと、例外「ページフォールト」が発生します。

例外によってカーネルが実行され、右の箱の中のように、二次記憶にある所望のページをページインします。（すべての実ページが使用中である場合は、まず、どれかをページアウトします）
ページフォールト例外から復帰すると、ユーザプロセスが再開してアクセスすべきメモリが存在するので、正しく実行できるようになっています。

ここで、「実ページが存在しない」は、ページテーブルエントリの「存在」情報を見ることでわかります。

仮想記憶のイメージ



仮想記憶の一つの見方は、プロセスに対する仮想アドレス空間が多数のページから構成されていて、そのうちの一部分が実メモリ上にキャッシュされているというものです。

キャッシュされなくてメモリから追い出されているものが二次記憶の中にバックアップとしてあります。

プロセスとアドレス空間

- プロセス … プログラムのコードおよび全ての変数、その他の状態
- 各プロセスは実行中、ユーザ空間上にロードされる
- 単一仮想記憶
  - システムで1つの仮想アドレス空間を作る
  - 主メモリの大きさにとらわれない記憶領域を形成することは可能
    仮想アドレス空間で、断片化などの問題は生じる
- 多重仮想記憶 … 一般的な仮想記憶の方式
  異なるプロセスに与えられる仮想アドレス空間が重複する方式
  - ユーザ空間はプロセス毎に割り当てられ、それぞれのアドレス範囲は同じ
  - 仮想記憶方式のオペレーティングシステムでは、ユーザプログラムのコードやデータは同じアドレスから開始されることが多い

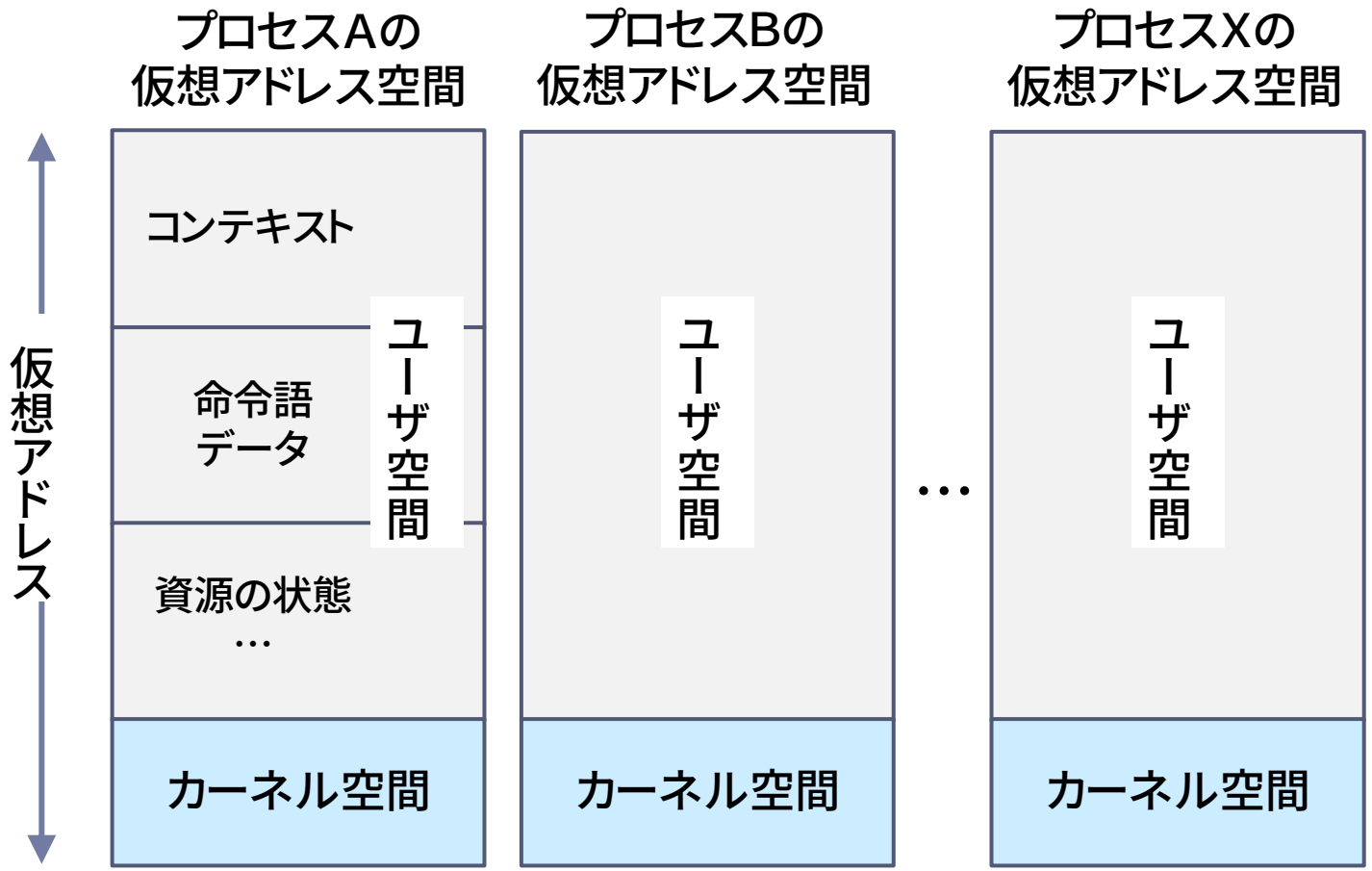
ここで、プロセスとアドレス空間の関係をまとめます。

システムで1つの仮想アドレス空間をもち、すべてのプログラムをそこに割り付けるという単一仮想記憶方式もありますが、ほとんどは、システム内に複数の仮想アドレス空間を形成する多重仮想記憶方式を用いています。

プロセスごとにそれぞれ別の仮想アドレス空間が与えられ、たとえば、仮想アドレスの100番地がどの実アドレスに対応するかは仮想アドレス空間ごとに個別に定義できます。

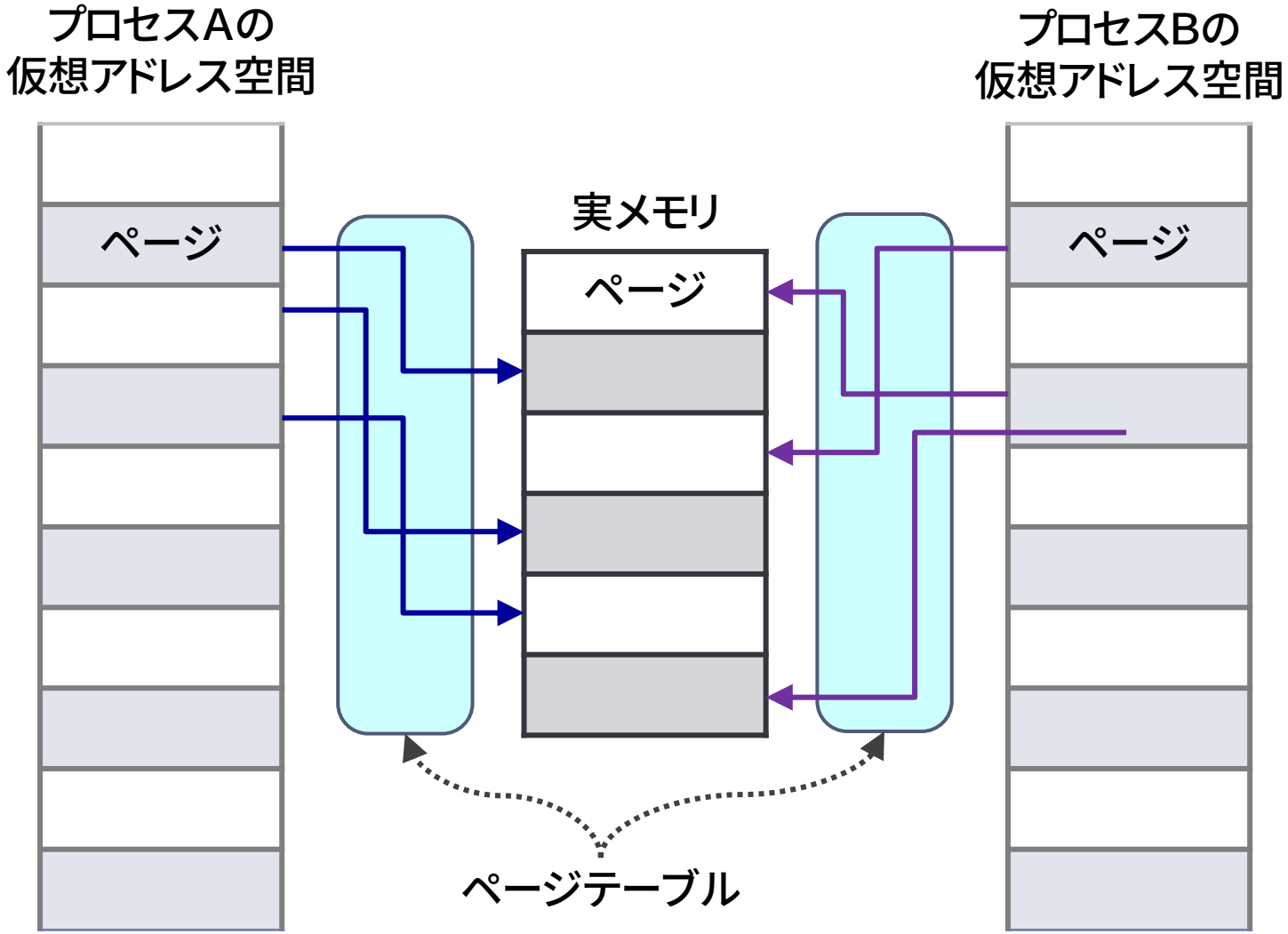
ユーザ空間とカーネル空間

- 一般に、1つの仮想アドレス空間をユーザ空間とカーネル空間に分けて使用する



一般には複数の仮想アドレス空間があり、それぞれユーザ空間とカーネル空間に分かれるというものです。

多重仮想記憶と実メモリの関係



p.24の図を複数プロセスに対応させて見てみますと、複数の仮想アドレス空間のそれぞれ一部分が実メモリにキャッシュされていることになります。
そして、ページテーブルがプロセスごとに存在します。

ページングによるメモリ保護

- プロセスと多重仮想アドレス空間
  - 多重アドレス空間で、プロセスごとに専用のアドレス空間をもつことにより、プロセス間でのメモリ保護が実現できる
  - プロセス間でページ単位で実メモリを共有することもできる
- ページ単位での保護と参照属性
  - 多くのプロセッサではページごとに「参照属性」を定義できる
    - 読込み属性
    - 書込み属性
    - 実行属性
    - 機械語を格納したページは通常書き込まれることはないので、「実行」属性と「読込み」属性にしておく
      →「書込み」でアクセスすると、例外が発生する

前のページでプロセスごとにページテーブルを持つ図を示していますが、このエントリで異なる実ページを持つようにすれば、他の空間のページを参照することはできません。
すなわち、プロセスに対して他プロセスからのメモリ保護ができます。
反対に、同じ実ページ番号を入れれば、プロセス間でそのページのメモリを共有できることになります。

また、ページ単位にそのページに対する操作の許可を設定することで不当な書き込みから保護するなどができます。ページテーブルにある管理情報(p.23)の中にそのような保護情報が設定されます。

教科書との対応

- p.23 の仮想記憶の処理の流れに関して、教科書 図10.9 の詳しい流れ図をあわせて見てください
- 変換ルックアサイドバッファ（TLB）については、次回に補足として紹介します

教科書での説明と記載箇所が違うところがありますので、読むときに注意してください。

第6回の課題

（1）仮想アドレス 20716 番地に対する 仮想ページ番号 と、ページ内オフセット値は、
ページサイズが ① 4KB（4096バイト）のとき、② 8KB（8192バイト）のとき、それぞれどうなるか

（2）ページングを実装する場合のオーバーヘッドにはどのようなものがあると考えられるか

今回の課題です。クラスウェブのレポートで提出してください。

期限は、5/25の午前中とします。

事後学習・事前学習

- 今回の講義資料に基づいて内容を振り返り、教科書などの該当箇所を読む
- 教科書第5章（5.1、5.2）に目を通す

今回の講義内容の振り返りと次回の準備をお願いします。