オペレーティングシステム(OS)

柴山　潔

12. 仮想メモリ (2)

- 仮想メモリの実現
- マッピング
- 置き換え

仮想メモリの実現 －仮想メモリ機構の概要－ (1)

■ 仮想メモリ機構：OSとハードウェアの分担によって，メインメモリ（実メモリ）とそのバックアップメモリでもあるファイル装置との連携（＝マッピング機能）を管理・制御

● プロセッサは，マシン命令中のオペランドによって，実メモリとは独立した“仮想メモリ”という一定サイズに固定した&巨大なメモリのアドレス（仮想アドレス）を指定可

- 仮想メモリ機構が仮想アドレス空間と実アドレス空間とをマッピング

  → ある1個の仮想アドレスから対応する1個の実アドレスを生成

- プロセッサは，仮想メモリ機構が生成する実アドレスによって実際のメインメモリ（実メモリ）にアクセス

仮想メモリの実現 －仮想メモリ機構の概要－ (2)

- メモリアクセスの属性である参照局所性を活用するために，仮想メモリと実メモリとのマッピングは一定サイズのブロックで
  - 実メモリ（実アドレス空間）は仮想メモリ（仮想アドレス空間）よりも小

    → ある時間に実アドレス空間にマッピング（＝領域割り付け）できる仮想アドレス空間はそのごく一部
  - 仮想メモリ機構が，「参照局所性が高い」ブロックを優先的に，実アドレス空間にマッピング

仮想メモリの実現 ー仮想メモリ機構の概要ー (3)

- 仮想メモリ(仮想アドレス空間)にあるアクセス対象(命令やデータ)は ファイル装置に格納
  - ファイル装置はメインメモリのバックアップ装置として機能
  - 仮想メモリ機構は,

仮想メモリ(仮想アドレス空間)と実メモリ(実アドレス空間)とのマッピング

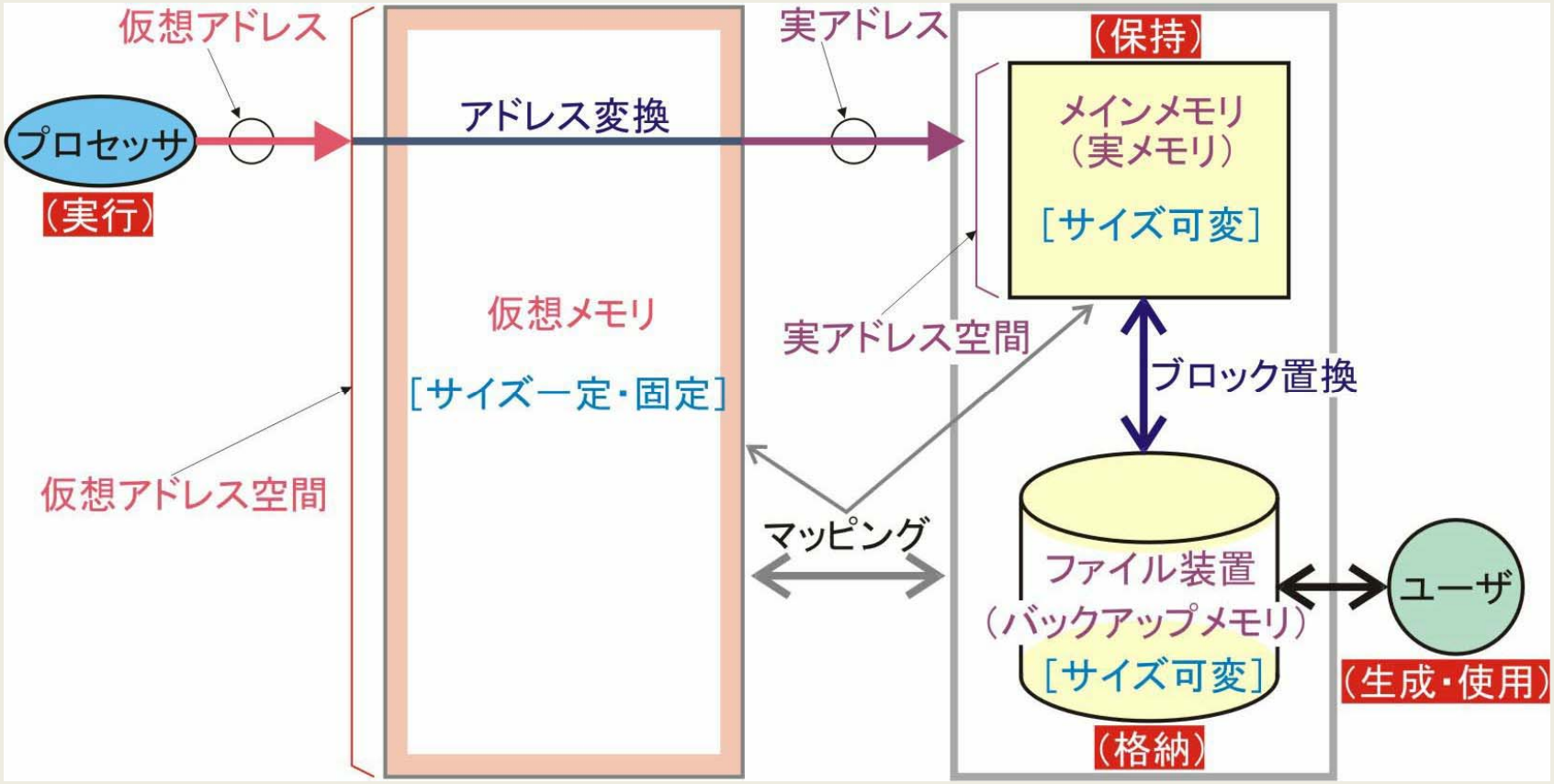
&

仮想メモリ(仮想アドレス空間)とその実体がバックアップしてあるファイル装置のアドレス空間とのマッピング

を統合して管理

→ 仮想アドレス空間と実アドレス空間とのマッピングを実現

仮想メモリ機構 ーハードウェア装置との関係（図説）ー



- プログラムやプロセスを

  プロセッサ→実行；メインメモリ→保持；ファイル装置→格納；ユーザ→生成・使用

仮想メモリ機構の動作 —メインメモリへのアクセス手順—

1. メモリオペランドから実効アドレス（＝アドレス指定モードで計算・生成するメモリアドレス）として生成する仮想アドレスを含むブロックが実メモリ上に（マッピングして）あるか/ないかをチェック
2. (a) ある → 仮想アドレス→実アドレス変換（＝ アドレス変換）
   (b) ない（＝ ページフォールト）

   → (1) 実メモリ上の不要ブロックをファイル装置（バックアップメモリ）へ追い出し，仮想アドレスで指定のアクセス対象の命令やデータを含むブロックを，ファイル装置から実メモリ上に取ってくる（入れ替え，＝ブロック置換，ページ置換，スワップ(swap)）

   (2) 仮想アドレス→実アドレス変換（＝ アドレス変換）

- ブロック置換時の「不要ブロックの決定戦略」＝ブロック置換アルゴリズム（ページ置換アルゴリズム）
- ページフォールトは代表的な割り込み要因
- ブロック置換は「ページフォールトを要因とする割り込み（＝ページフォールト割り込み）に対するOSによる割り込み処理機能」

多重仮想アドレス空間 (1)

■ “OSで管理する仮想アドレス空間が単一/複数”による仮想メモリ方式の分類

(A) 単一仮想アドレス空間：単一の仮想アドレス空間を複数プロセスで共有利用
→ OSが管理する仮想アドレス空間は唯一

- ○ 機能の実現は簡単
- × 仮想アドレス空間での複数プロセス間相互のメモリ保護や動的リロケーション機能が必要

(B) 多重仮想アドレス空間：プロセスなどの論理的なプログラムブロックごとに仮想アドレス空間を割り付け
→ OSが管理する仮想アドレス空間は複数個で多数

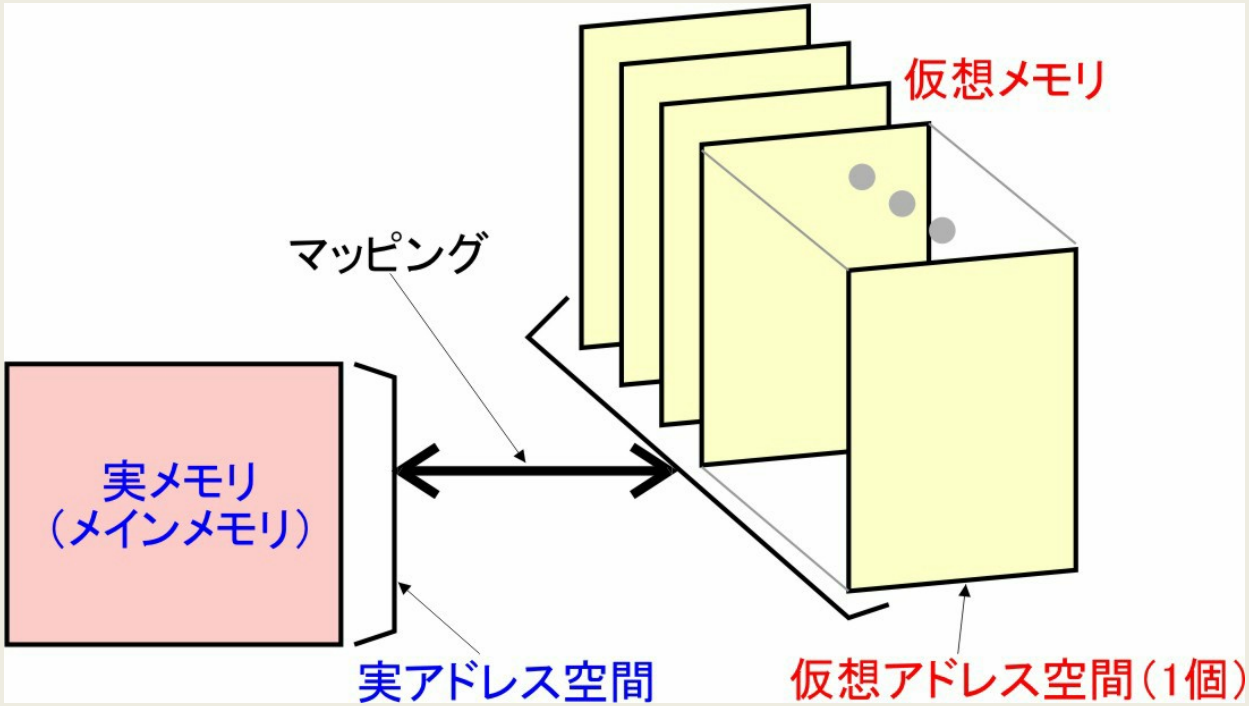
- ○ プロセスごとに個別の論理アドレス空間
- ○ 仮想メモリの効果“仮想アドレス空間を実アドレス空間から独立して設定可”をマルチタスキングでさらに活用可

多重仮想アドレス空間 (2)

■ 多重仮想アドレス空間の適用によるOSに対する具体的効果

- プロセスごとに論理アドレス空間を割り付け･管理可
- 個々のプロセスのアドレス空間が独立 → 複数プロセス間相互のメモリ保護が容易

➢ 現代のOSのほとんどは多重仮想アドレス空間方式を採用



【まとめ】仮想メモリ機構でのOSの役割

- 仮想メモリ機構はハードウェアとソフトウェア（特に，**OS**）とが機能分担して構成
  - 仮想メモリ機構は，アクセス対象の仮想アドレスを含むブロックが実メモリ上に　(A) “ある”場合：アドレス変換だけ；(B) “ない”場合：ブロック置換 → アドレス変換
  - アドレス変換もブロック置換も，「**OS**とハードウェアとの機能分担（ ＝OSの管理・制御下で動作するハードウェア機構）」によって，実現

- アドレス変換時に発生する（可能性がある）ページフォールト割り込みは，仮想メモリ機構における“ハードウェア装置→**OS**通信”
  - ページフォールト割り込みは「ハードウェアのアドレス変換機構から**OS**への“ブロック置換”の依頼」
  - **OS**がページフォールト割り込みを受け付けると，その割り込み処理において，「ブロック置換アルゴリズムによる不要ブロックの選択やブロック置換そのものの管理や制御」を実行

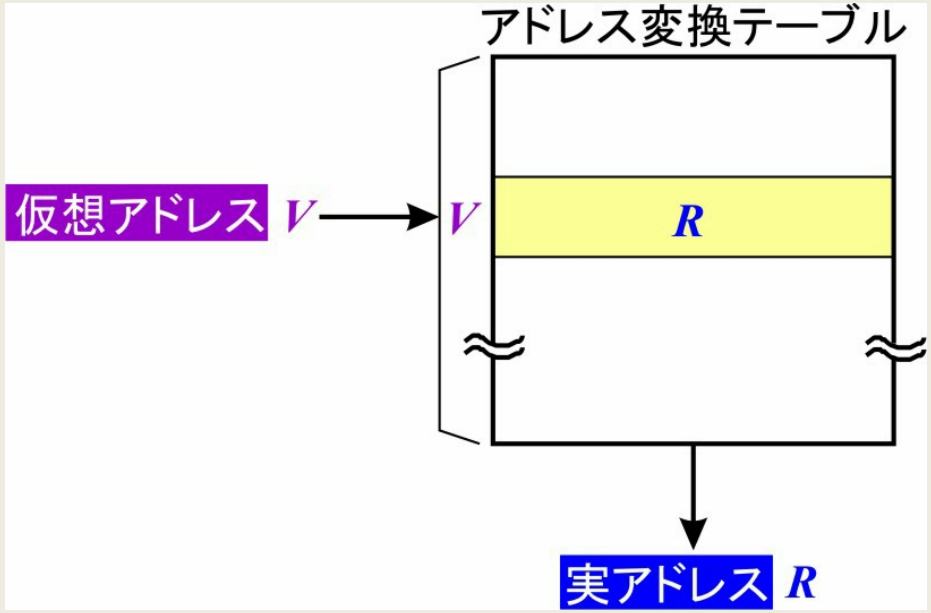
- 仮想メモリは，その主要な機能を**OS**で実現することによって，メインメモリ性能の空間的改善を図るメモリアーキテクチャ
  - 一方，**OS**は，“仮想メモリ機構”というハードウェアの性能を活用することによって，メインメモリの時間的性能も維持

アドレス変換テーブル

＝ マッピングテーブル(mapping table)

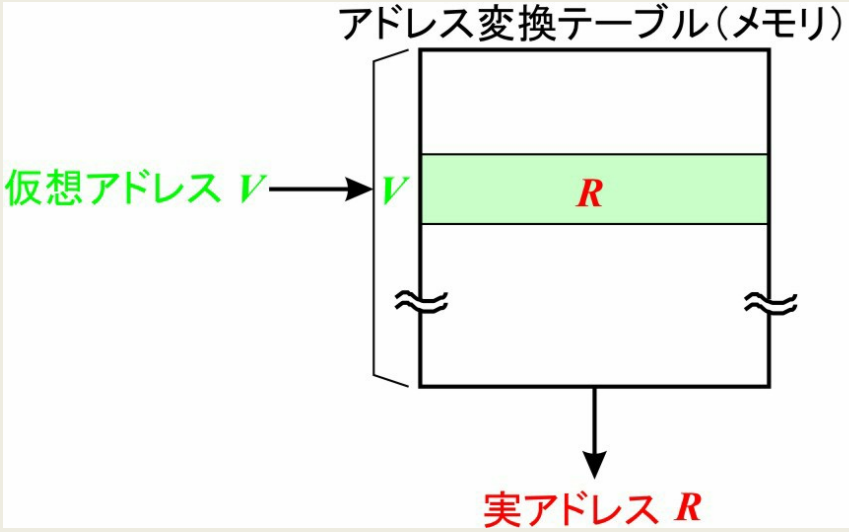
- 仮想アドレスと実アドレスのマッピングを記述
- 仮想アドレス *V* → 実アドレス *R* のアドレス変換の機能
  - 「アドレス変換テーブルを引く」操作が「当該仮想アドレス（のマッピング）が実メモリ上にある/ないのチェック」操作も兼ねる
- 通常はメインメモリ内に実装

  ↑（理由）
  - マッピングの変更ごとに動的書き換えが必要
  - サイズが大

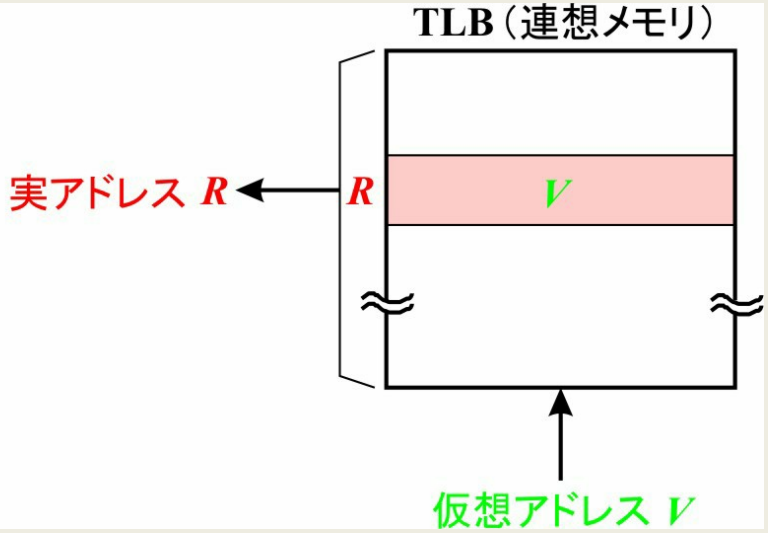


メモリと連想メモリ —仮想メモリ機構での例—（参考）

■ メモリによる構成例



■ 連想メモリによる構成例



● アドレスによって格納内容を引く

● 格納内容によってアドレスを引く

仮想アドレス *V* → 実アドレス *R*

● アドレス変換バッファ(**TLB**; **T**ranslation **L**ookaside **B**uffer, **T**able **L**ook-up **B**uffer) : 参照頻度が高い「仮想アドレスと実アドレスとのマッピング」だけを保持する専用ハードウェア機構（＝連想メモリ）

マッピングの分類

- 「仮想メモリ⇔実メモリ(メインメモリ)間のマッピング単位の決定方法」による分類

■ マッピング単位 ブロック ＝ (1) 参照局所性が高い仮想メモリ部分の実メモリ(メインメモリ)領域への割り付け単位;(2) 実メモリ(メインメモリ)⇔バックアップメモリ(ファイル装置)間のデータ転送単位

(A) ページング(paging)

- ページ(page)という一定サイズに固定したブロック(単位)でマッピング
  - ➢ プログラム，データ，プロセスの「論理的な意味」は無視 → 機械的/物理的に，固定長(一定)サイズのページ単位に区切る ＝ 固定長領域割り付け

(B) セグメンテーション(segmentation)

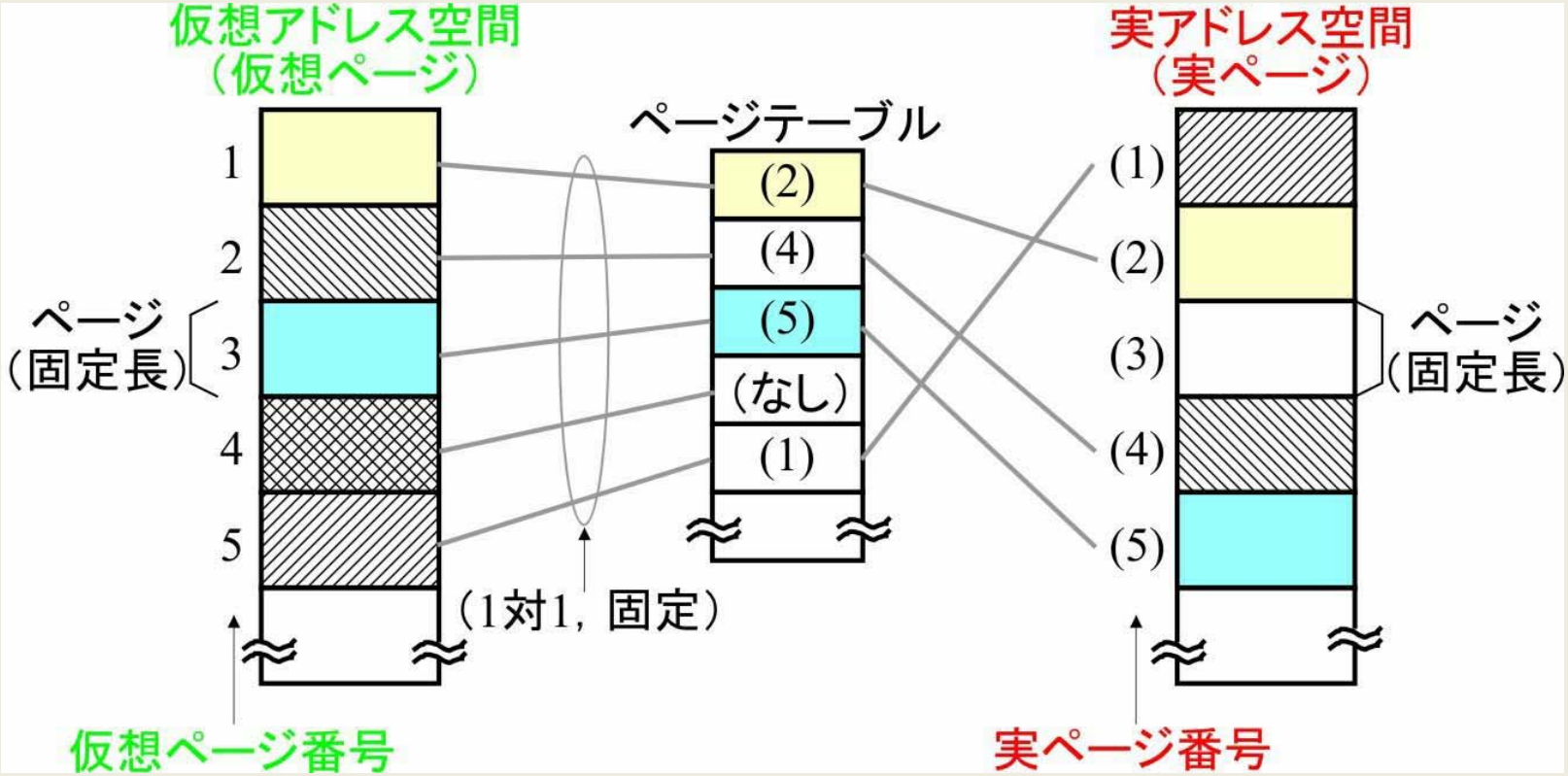
- 1個のプログラムやプロセス，一連(一群)のデータという「論理的な意味」をもつブロック(＝セグメント(segment))でマッピング
  - ➢ セグメントは，プログラム/プロセスやデータブロックという「論理的な意味」を考慮した論理的単位 → セグメントのサイズは可変長 ＝ 可変長領域割り付け

(C) ページセグメンテーション(paged segmentation)

- (A)と(B)を融合したマッピング

ページング

- 仮想アドレス空間も実アドレス空間も一定&固定長のページサイズで分割
- ページ単位で仮想アドレス空間上のページ（仮想ページ，論理ページ）と実アドレス空間上のページ（実ページ，物理ページ）とをマッピング



ページングの特徴 －OSによるメモリ管理の観点－

長所

(1) マッピング単位が一定・固定長ページ → メインメモリ（実メモリ）やファイル装置（バックアップメモリ）の管理が簡単，メインメモリでの外部フラグメンテーションの発生はまれ，メインメモリの使用効率は常時良好

(2) メインメモリ（実メモリ）サイズより大きなプロセス/プログラム/データも，複数ページに分割して，必要な使用する仮想ページだけを動的入れ替えで実メモリ上にマッピング可能

短所

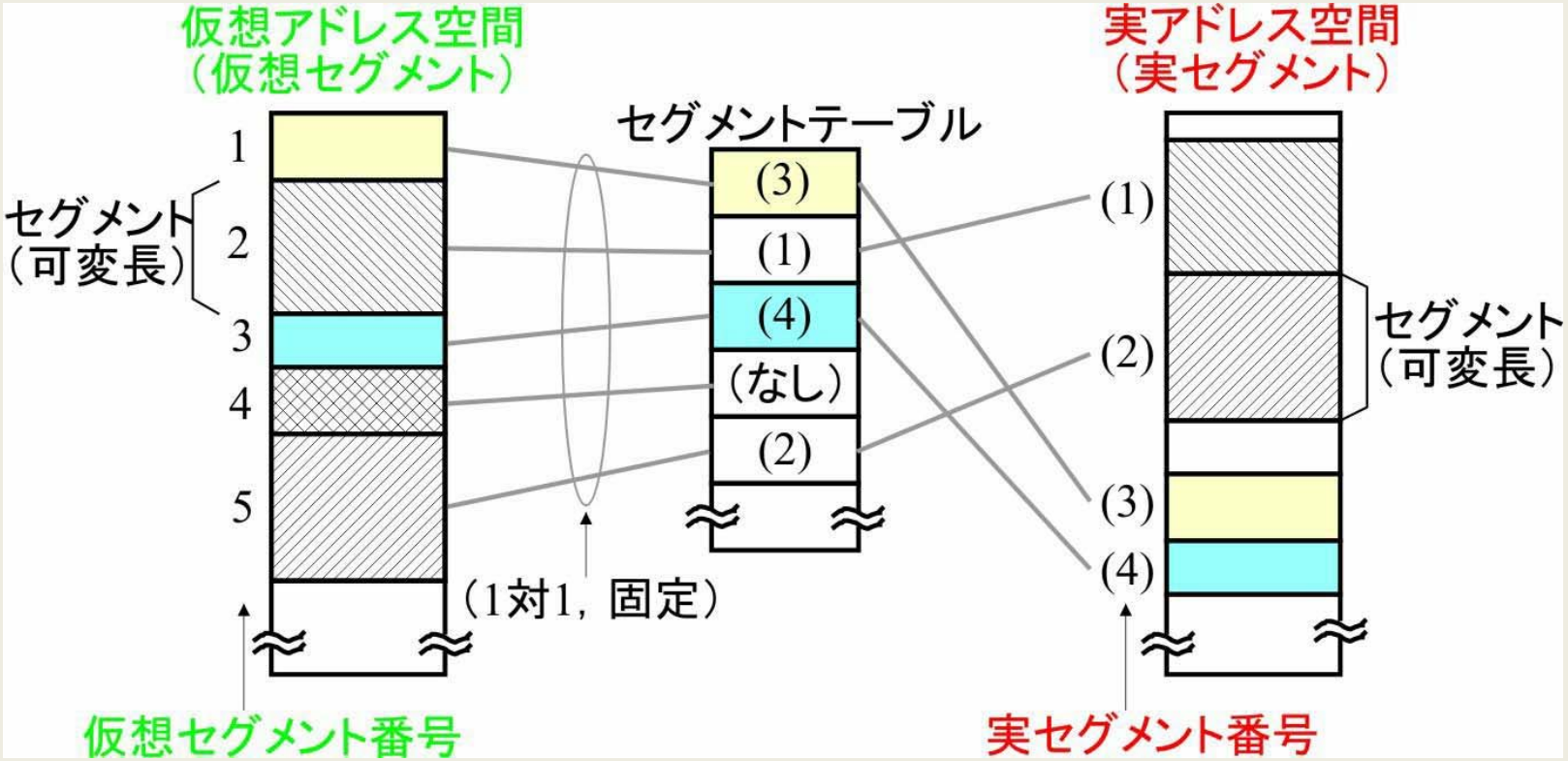
(3) プログラム/データの「論理的な意味」を無視して機械的/物理的にページに分割かつ不連続で独立にマッピング → 「論理的な意味や論理的単位そのものが備える参照局所性を活用する」が困難

- ◆ 実アドレス空間上でのプロセス/プログラム/データの属性管理やリンク（link; 関係付け，連係）などを無視する分割なので，参照局所性を適用/利用する方法が複雑に

(4) 「ページサイズが一定・固定」&「ページ単位マッピング」 → ページサイズよりも格段に小さいプロセス/プログラム/データをマッピングした場合，ページ内に大きな未使用領域（＝ 内部フラグメンテーション）の無駄

セグメンテーション

- セグメント（部分/区分/文節/一区切り）単位で実アドレス空間（実セグメント，物理セグメント）と仮想アドレス空間（仮想セグメント，論理セグメント）とをマッピング



セグメンテーションの特徴 －OSによるメモリ管理の観点－

長所

(1) セグメント（部分/区分/文節/一区切り）は「論理的な意味」をもつ単位 → セグメントテーブルに記述しておくセグメントの属性（例：アクセス権；命令/データの種別；データ型；他のセグメントとの関係；共用/非共用の区別 ）をアクセス制御時に使用可，実行時の動的プログラム間リンク（＝参照関係の解決）や共用ライブラリの構成が容易

(2) セグメント単位での割り付け → メインメモリ内に領域を確保できれば，内部フラグメンテーションは不発生

短所

(3) セグメントサイズが可変・自由 →「メインメモリ（実メモリ）やファイル装置（バックアップメモリ）での領域管理」および「実メモリと仮想メモリとのマッピングの管理」が複雑，メインメモリ（実メモリ）での外部フラグメンテーションが発生しやすい → メインメモリの利用効率は悪

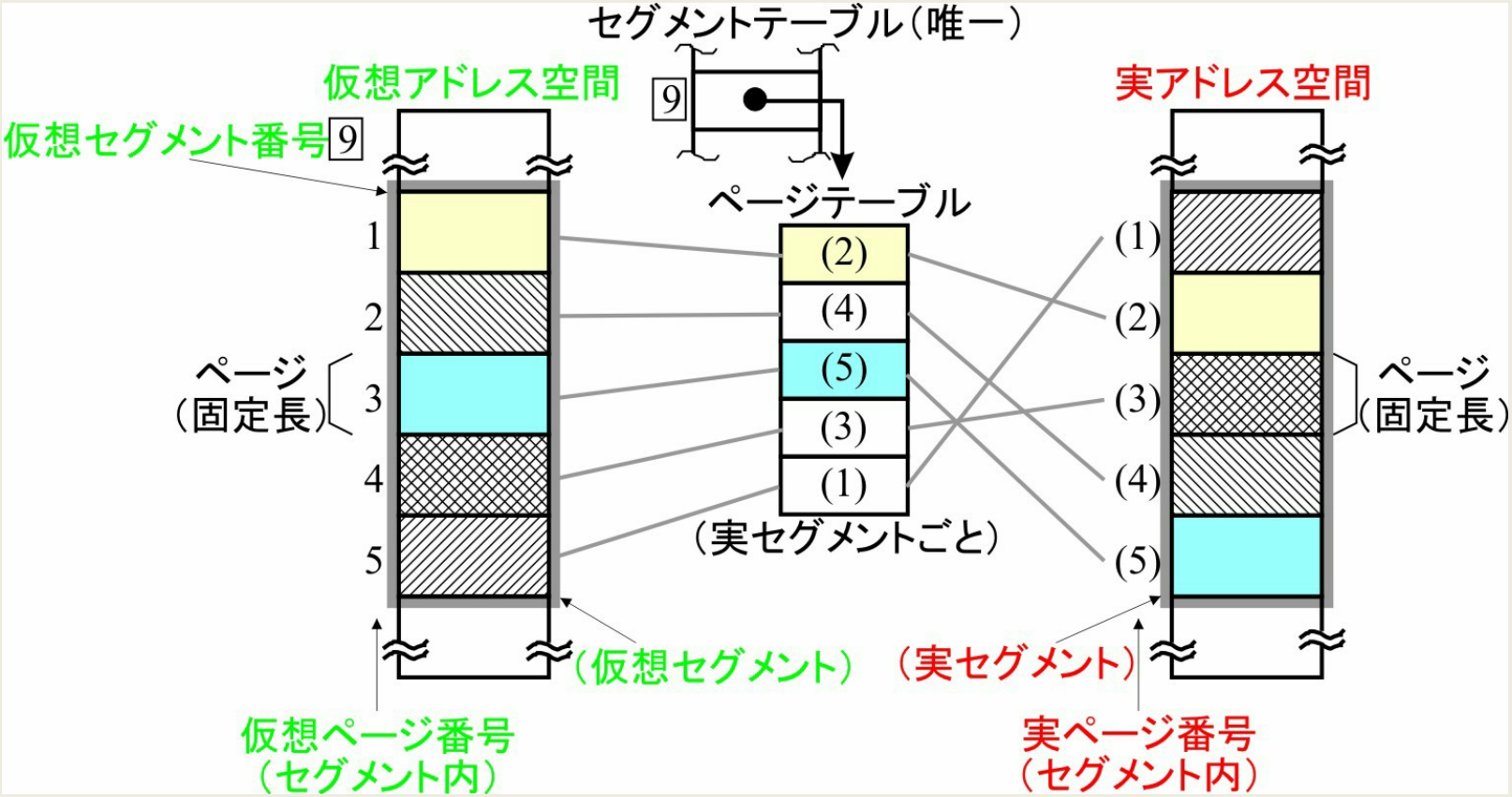
(4) メインメモリサイズより大の仮想空間をセグメントとして割り付け不可 → その場合にセグメンテーションの長所(1)(2)を喪失

ページセグメンテーション

1. 全体ではセグメント単位でのマッピング［セグメンテーション］

＋

2. 各セグメント内ではページ単位でのマッピング［ページング］



ページセグメンテーションの実現

- 仮想アドレス空間も実アドレス空間も一定･固定長のページサイズで分割

■ マッピング

1. セグメントテーブルによって，セグメント単位で，仮想セグメントと実セグメントとをマッピング
   - セグメントはひとまとまりだが，全体サイズはページ単位で可変･自由
2. ページテーブルによって，ページ単位で，仮想/実セグメントのそれぞれのセグメント内のページどうしをマッピング
   - ページテーブルは各実セグメントごとに装備

■ アドレス変換

1. セグメントテーブルによって，仮想セグメント番号→ページテーブル（実セグメントごと）アドレス
2. ページテーブルによって，仮想ページ番号→実ページ番号

● 仮想セグメントをページ単位で分割，その各ページを実セグメント内のページに独立して不連続にページングでマッピングするセグメンテーション

ページセグメンテーションの特徴 －OSによるメモリ管理の観点－

長所

(1) ページングとセグメンテーションの両方式の長所を融合 → 最初のマッピングはセグメンテーションで，セグメントの「論理的な意味」を活用可 & 後のマッピングは，ページングで，実メモリ管理が簡単かつ利用効率も優

短所

(2) セグメントテーブルとページテーブルの2種類のアドレステーブルが必要 → テーブル全体のサイズが大 & 2段階のアドレス変換（テーブルの検索）時間も大

- ページテーブルを不要にする改良型ページセグメンテーション方式によって，(2)の短所は隠ぺい & (1)の長所が生きる

■ ページセグメンテーションは現代のOSの仮想メモリ機構を支える代表的マッピング方式

【まとめ】仮想メモリへのプロセス割り付けの実際

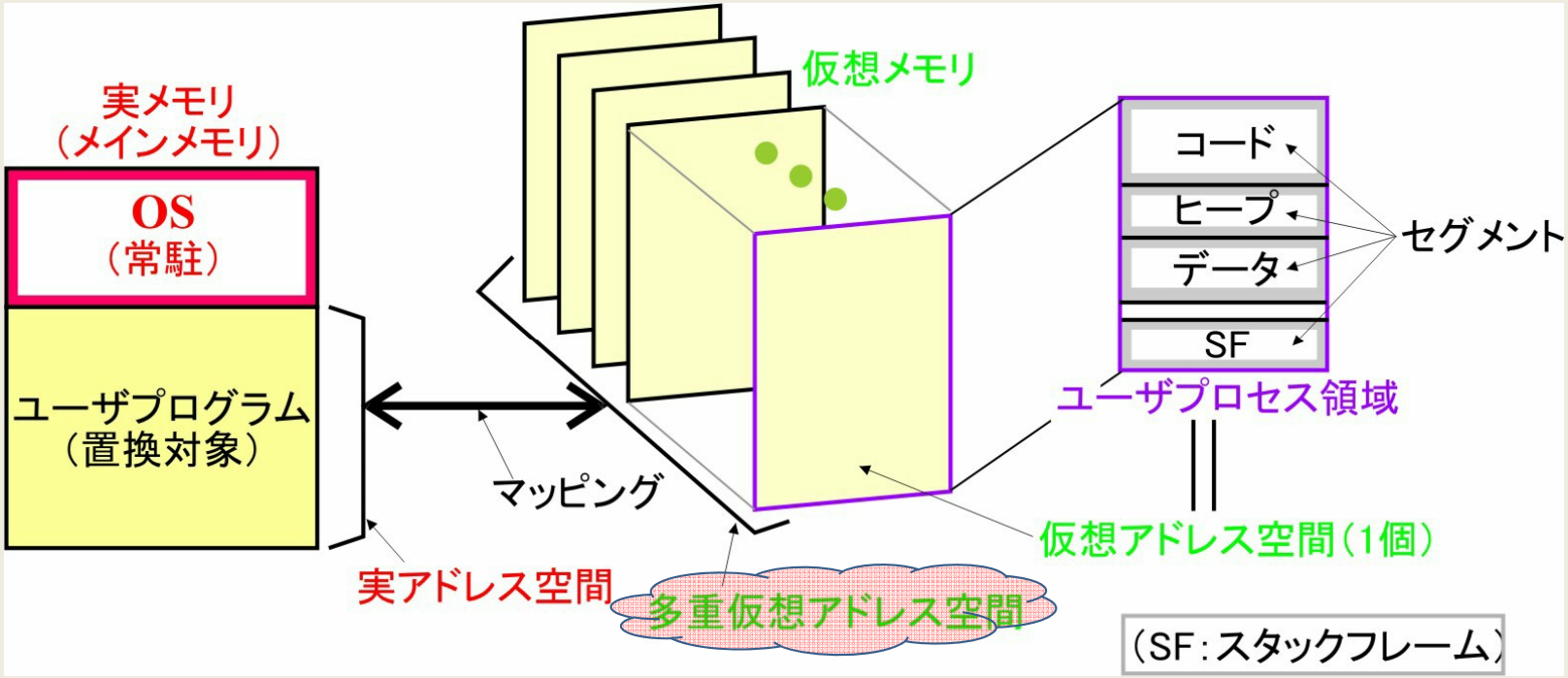
（現代のOS）多重仮想アドレス空間 & ページセグメンテーション

■ プロセス割り付けの方針
- ユーザプロセス領域ごとに，1個の独立した仮想アドレス空間
- ユーザプロセス領域を構成するコード/データ/ヒープ/スタックフレームの各領域はそれぞれ独立したセグメント
- 実メモリ（メインメモリ）へマッピングする（＝割り付ける）ユーザプロセス領域（＝セグメント）のために，バックアップメモリ（ファイル装置）上に，一定サイズ領域（＝スワップ領域）をあらかじめ確保
- OS自身は，仮想メモリ（ブロック置換）の対象外，実メモリ（メインメモリ）に一定サイズ領域を直接確保&その領域に常駐

■ プロセス割り付けの実際
- プロセッサが実行するプログラム（命令やデータ）は，実行時に，OSが，プロセスとして，メインメモリに割り付け
  - 仮想メモリ機構が稼働している場合，プロセス割り付けにおいて，仮想メモリ機構が，ブロック置換によって，必要プロセスをファイル装置から読み出し，メインメモリに置く（＝割り付ける）
  - 一方，「不要」と判定可の（＝参照局所性が低い）プロセスは仮想メモリ機構が，ブロック置換によって，メインメモリからファイル装置へ追い出す

仮想メモリへのプロセス割り付け（図説）

（現代のOS）多重仮想アドレス空間 & ページセグメンテーション



【まとめ】ページフォールトとブロック置換

■ ページフォールト割り込み

- ページ単位のマッピング（＝ページングやページセグメンテーション）において，

➢ アクセスを要求した仮想ページがメインメモリ（実メモリ）になし

➢ （実際は）ページテーブルにアクセス対象の仮想ページの登録なし　＝ ページフォールト

の場合に生じる（をOSに通知する）割り込み

↓

■ ページフォールト割り込みに対する割り込み処理

＝ ブロック置換（＝ ページ置換）

1. 仮想ページ（←ファイル装置内）⇔実ページ（←メインメモリ上）の置換（＝スワップ(swap)）
   ➢ メインメモリからのページ（スワップ）アウト（page(swap)-out; 追い出し）&メインメモリへのページ（スワップ）イン（page(swap)-in; 読み込み）
2. ページテーブルの書き換え

（上級）

スラッシング(thrashing)

- 実メモリ容量（実アドレス空間サイズ）が小 → 実メモリ上での参照局所性の使用が不可能 → ブロック置換が頻発

- プロセッサがブロック置換管理（OS機能のひとつ）にかかりきり → ユーザプロセスやブロック置換管理以外のOS機能の実行が実質上不可能
- スラッシングはオーバヘッド → 仮想メモリの効果のためには防止（＝充分な容量の実メモリの装備）が必須

（上級）

ページ置換アルゴリズム (1)

- ページ置換では，「使用頻度が高い（参照局所性が高い）ページをメインメモリ（実メモリ）で保持」が目標
  - 「使用頻度（＝アクセス可能性＝参照局所性）が低い実ページをスワップアウト」が有効

［定義3.4］ **ページ置換アルゴリズム**

- ページ置換時に「メインメモリに置いてある実ページのどれをファイル装置にスワップアウトするか」を決定する戦略（アルゴリズム）

- ページフォールトの発生で，**OS**はページ置換アルゴリズムにしたがってスワップアウトするページ（＝使用頻度＝アクセス可能性＝参照局所性が低いページ）を決定
  - ページ置換アルゴリズムが不適切 → （適度なサイズの実メモリ容量が確保できても）スラッシングを引き起こす

（上級）

ページ置換アルゴリズム (2)

■ ページ置換アルゴリズムの評価指標

(a) ライフタイム(life-time) : ページフォールト間の平均時間間隔（＝ページの平均実メモリ滞在時間）

- 長 → アルゴリズム良

(b) ページフォールト率 : ライフタイムの逆数

- 低 → アルゴリズム良

（上級）

代表的なページ置換アルゴリズム (1)

(1) LRU (Least Recently Used)：最後のアクセス時刻が最古のページをスワップアウト対象に
- ○ 時間的参照局所性を自然に反映→ ページフォールト率が低
- × アクセス履歴を記録･比較するページ置換管理機構が複雑 → ページ置換時間は長

(2) FIFO (First In First Out)：メインメモリに一番最初（＝最古）にスワップインしているページを最優先のスワップアウト対象に
- △ ページ置換機構の実装コストは(1)LRUと(3)ランダムとの中間
- △ アクセスや参照はノーチェック→ 参照局所性の反映度は(1)LRUよりも低

(3) ランダム(random)：任意/無作為にスワップアウトするページを決定
- ○ 実現/実装は簡単
- × 参照局所性の考慮は皆無（＝アルゴリズム/戦略とはいえない）

（上級）

代表的なページ置換アルゴリズム (2)

(4) ワーキングセット(working-set)：ワーキングセット（定義3.5，下記）ではないページを優先すべきスワップアウト対象に

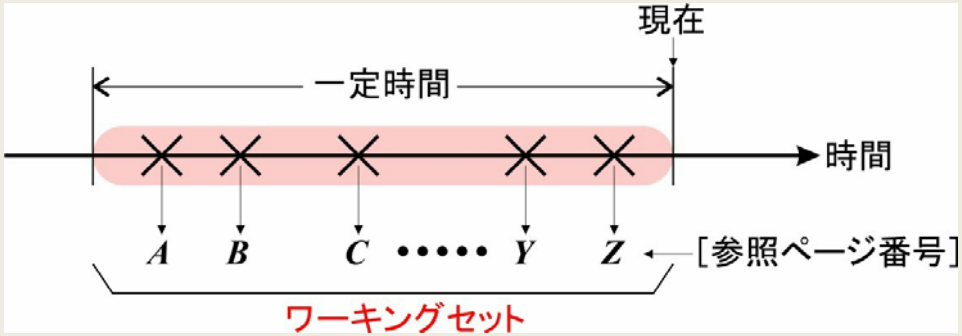
- 「ワーキングセットではない」ページから，(1)～(3)などでスワップアウト対象を決定

○ 時間的参照局所性を正確に反映

× 機構は大規模&複雑

［定義3.5］ ワーキングセット

- あるプログラムを実行している時刻から一定時間をさかのぼる過去の間に，当該プログラムが参照したページを要素とする集合 ＝ 参照局所性が高いページの集合



（上級）

代表的なページ置換アルゴリズム (3)

（実際）

- ページ置換アルゴリズムよりも，アクセスや参照対象のプログラム/データブロックそのものの参照局所性がページフォールト率を左右
  - → どのアクセスや参照の対象（プログラム/データ）に対してもページフォールト率が低くなるような最適ページ置換アルゴリズムの設計/選定は難

（上級）

ページ置換のタイミング (1)

■「いつ/どのタイミングでページを仮想アドレス空間から実アドレス空間に読み込む（＝スワップインする）か？」によるページングの分類

(A) **デマンドページング**（demand paging; 要求時ページング）：プログラム（実行中プロセス）が実行時（動的）に参照やアクセスを要求（＝デマンド）するページをそのたびに読み込み

- **OS**とハードウェアの機能分担（**OS**機能をハードウェア機構が支援）による動的ページング

○ 必要（＝要求する）ページだけ読み込み → 無駄がない

× プロセス実行の開始（＝初めての実行中状態への遷移）時にページフォールトが集中発生 → これによるページ置換処理の間は「先に読み込まれるページを使用するプロセスは未実行で実行可能状態のまま待つ」という時間的&空間的冗長性あり

→ 適切なページ置換アルゴリズムの選択が重要

（上級）

ページ置換のタイミング (2)

(B) プリページング（pre-paging; 先行ページング，予測ページング）：OSがあらかじめ「アクセスや参照がある」と予測するページ（通常は 複数ページ）を実行前（静的）にまとめて読み込み

- OS とコンパイラの機能分担（OS機能をコンパイラ機能が支援）による静的ページング

○ 予測の的中率が高 → 複数ページを一度に読み込むブロック転送が適用できるので，読み出し回数とページ当たりの平均読み出し時間は減 → 高速化

× 不要ページを読み込みの可能性あり

× 予測の的中率が低 → 予測に要するオーバヘッドが顕在化
→ 予測に要するオーバヘッドを上回る（ページ置換そのものの）高速化が必須

↓

- （現代）メインメモリの実装コストが低 → 大容量メインメモリを実装 → 余分&冗長なページも読み込み可 → 予測がはずれた(＝読み込みページの不使用)場合の影響は少

↓

● 現代では，プリページングを採用するOSが多

（上級）

OSによるメモリ保護機能

- メインメモリ領域に保持するプロセス（プログラム）の個々について，アクセス権（定義3.6）にしたがって，相互に保護し合う機能
  - 「OSがアクセス権を管理する」によって実現

［定義3.6］ アクセス権

- 「当該プロセスが，他プロセスに対して，ある アクセス形態 を許可するか禁止するか」に関する取り決め
  - アクセス形態 の例：読み出し，書き込み，（命令としての）実行
  - アクセス権はプロセス（プログラム）ごとに設定

- ユーザプロセス（ユーザプログラム）の実行中に，アクセス権に違反する不正アクセス（＝メモリ保護違反）があれば，それを要因とする割り込みが発生
  - ユーザプログラムの実行（＝ユーザ状態）からOSの実行（＝カーネル状態）に切り替え&メモリ保護違反に対する割り込み処理を実行

（上級）

OSによるメモリ保護 —例：仮想メモリ機構への組み込み—

■ ページング/ページセグメンテーションのマッピング方式を採る仮想メモリ機構に組み込む例

(1) 物理ページごとにメモリ保護に関する情報を保持
(2) ページテーブル（アドレス変換テーブル）中にメモリ保護に関する情報を付加

- (1)(2)ともに仮想メモリ機構の一部として実現可
- 特に，多重仮想アドレス空間に(2)を適用 → プロセス個々の仮想アドレス空間（論理アドレス空間）ごとに独立したメモリ保護機能やアクセス権を設定･運用可