オペレーティングシステム

3章 メモリ管理

3.2節 仮想メモリーメインメモリの隠ぺいー（後半）

大阪大学大学院情報科学研究科
村田正幸
murata@ist.osaka-u.ac.jp
http://www.anarg.jp/

3.2 仮想メモリーメインメモリの隠ぺい―
3.2.3 マッピング

[1] アドレス変換テーブル

- 仮想アドレス*V*から実アドレス*R*へのアドレス変換
  - 仮想アドレス*V*によってアドレス変換テーブルを引き、その内容（エントリ、事項）である実アドレス*R*を得る
  - アドレス変換テーブルを引く操作が当該仮想アドレスが実メモリ上にあるかないかのチェック操作も兼ねる
- メインメモリ上に構成
  - マッピングの変更ごとに動的な書き換えが必要となる
  - サイズが大きい
- 以下の操作は高速処理が要件となるので、通常は、動的アドレス変換機構 (DAT) によって実現
  - 「当該仮想アドレスが実メモリ上にあるかないかのチェック」操作
  - 「アドレス変換テーブルを引く」操作

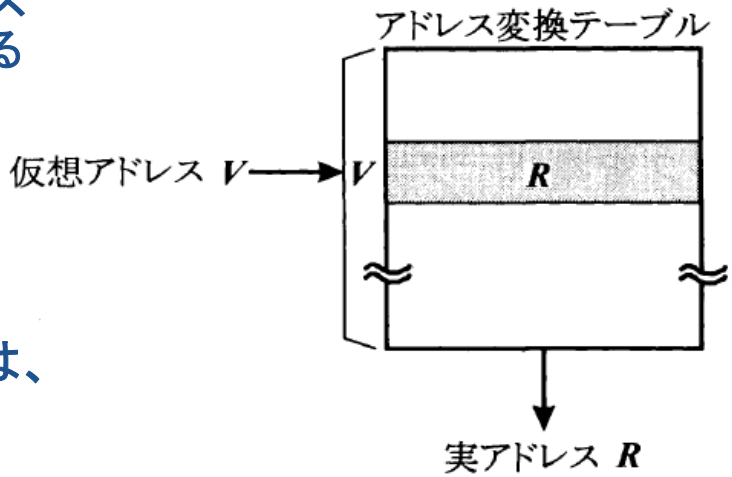
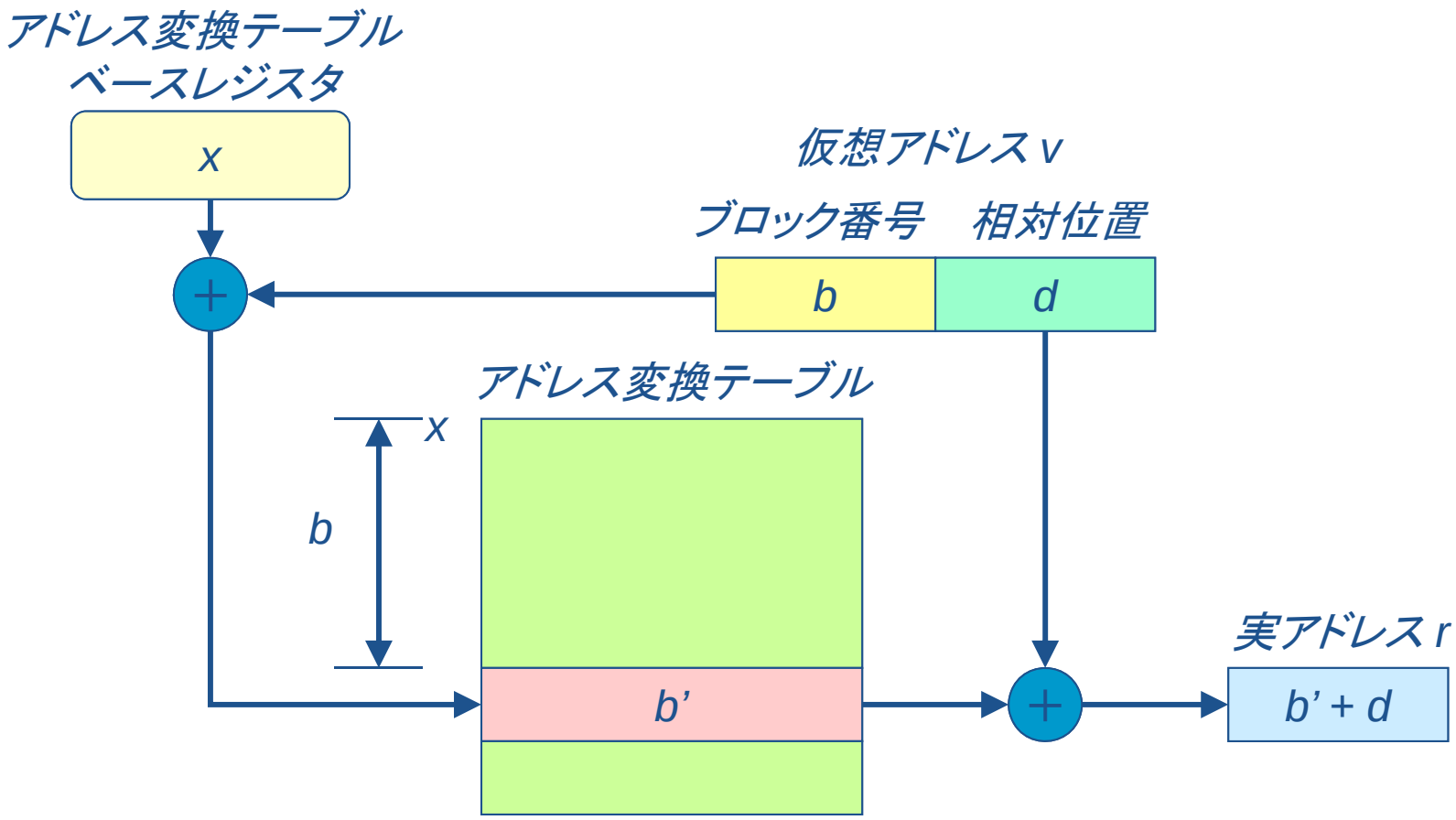
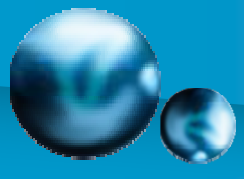


図 3.12　アドレス変換

動的アドレス変換（DAT）



実アドレス＝アドレス変換テーブル[ブロック番号]＋相対位置



3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

[2] 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
  - 仮想メモリの一部(コピー)を実メモリ(メインメモリ)領域へ割り付ける機能
- 単位
  - 仮想メモリのうちの参照局所性の高い部分の実メモリ(メインメモリ)領域への割り付け単位
  - 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
- 分類

(A) ページング(paging)

- ページという一定サイズに固定したブロック(単位)でマッピングする
- プログラム(データ、プロセス)は、それらがもつ「論理的な意味」は無視して、機械的また物理的に、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の例

(B) セグメンテーション(segmentation)

- 1 個のプログラムやプロセスあるいは一連(一群)のデータといった「論理的な意味」を持つブロック(セグメントという)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」を考慮した論理的な単位で構成するので、そのサイズは可変
- 可変長領域割り付けの例

(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

3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

[3] ページング

- 仮想アドレス空間も実アドレス空間も一定かつ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（仮想ページまたは論理ページ）と実アドレス空間上のページ（実ページまたは物理ページ）とをマッピングする
- 仮想アドレス空間には仮想ページごとに仮想ページ番号を、実アドレス空間には実ページごとに実ページ番号を、それぞれ振る
- 仮想ページと実ページのマッピングはページテーブルと呼ぶアドレス変換テーブルに記述しておく
  - 仮想ページから実ページへのアドレス変換はページテーブルを引く操作となる

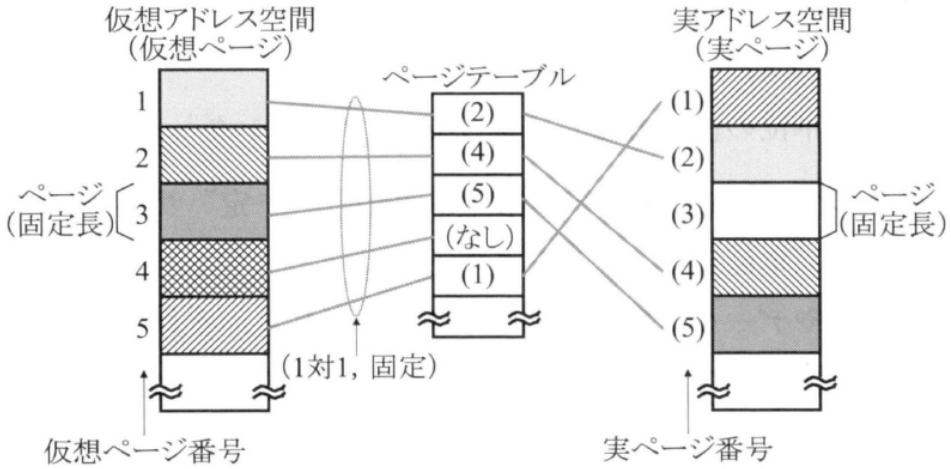
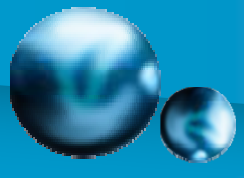


図 3.14 ページングによるマッピング（例）



3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

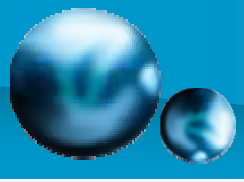
[3] ページング

- 長所
  1. マッピング単位が一定かつ固定長のページ

     →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ほとんどなくなり、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（プログラムやデータ）も、実メモリ上にマッピングできる

     ←複数ページに分割して一部の必要なあるいは使用する仮想ページだけを動的に入れ替え



3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

[3] ページング（続き）

- 短所

3. プロセス（プログラムやデータ）がもつ「論理的な意味」を無視して、機械的また物理的にページに分割して、かつ、それらを不連続で、独立にマッピングする

→論理的な意味や論理的単位そのものが備えている参照局所性を活用することが困難になる

→元の論理的な意味や論理的単位で備えていた参照局所性も分割されてしまうので、その参照局所性を適用あるいは利用する方法は格段に複雑になる

←実アドレス空間上でのプロセスの属性管理やリンク（関係付け、連係）などの論理的な意味や論理的単位を無視して、それらを機械的また物理的に分割するため

4. ページサイズが固定であり、また、ページ単位でマッピングする

→内部フラグメンテーションが発生しやすい

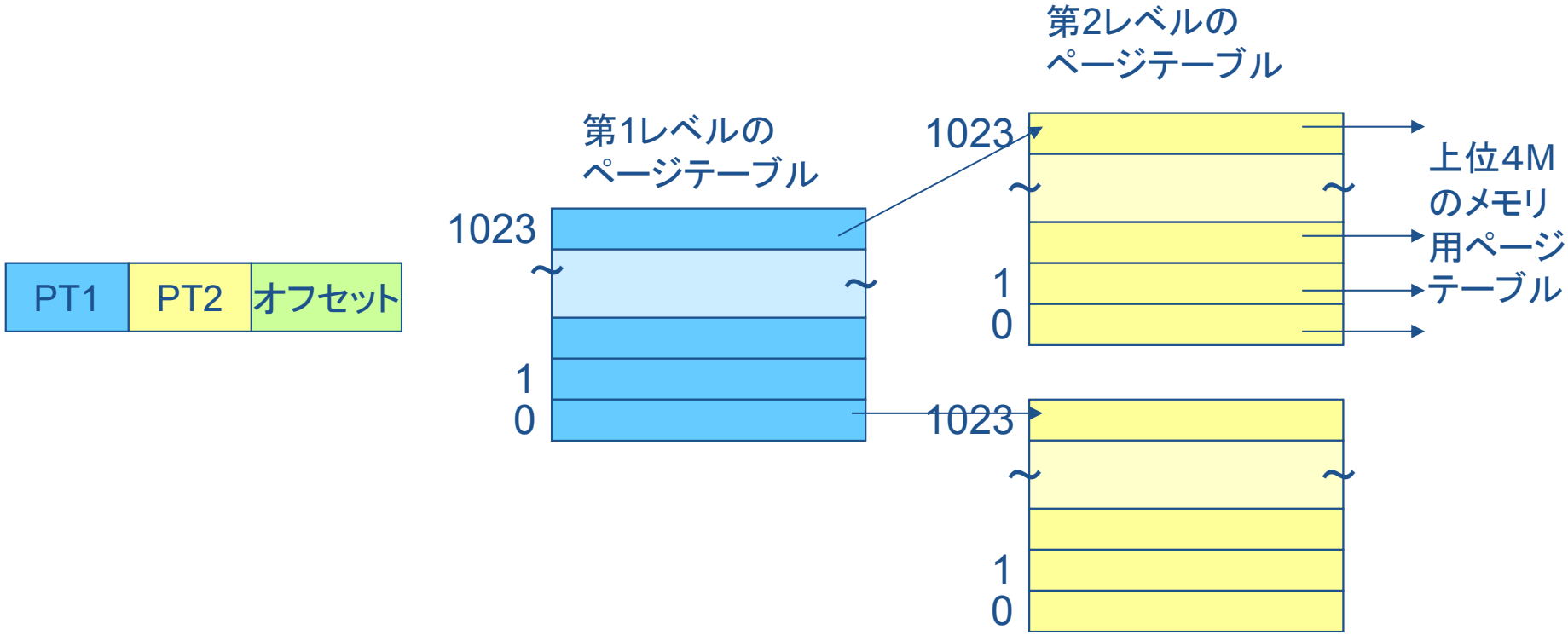
→特に、ページサイズよりも格段に小さいプロセスをマッピングした場合、そのページ内に残っている大きな未使用領域を他のプロセスにマッピングできないという無駄が生じる

ページテーブルの構成

- メインメモリをページ枠 (Page Frame) に分割
  - ページ枠への割り当てはメインメモリ上のページテーブルで管理
  - ページ(ページ枠)の大きさ:4096バイトや8192バイト
    - ページが小さい→テーブルが大きくなって、検索時間が増大
    - ページが大きい→無駄な領域(内部断片化)が大きくなる
- ページテーブルの構成
  - 32 bit 仮想アドレス⇒20 bit ページ番号, 12 bitオフセット⇒1エントリ 32 bit (実ページ番号＋属性)
  - 2^{20} x 4 x n = 4 x *n* MB (*n*: 仮想アドレス空間の数≒同時に存在するプロセス数)

ページテーブルの巨大化への対応

- 多段ページテーブル、TLBなど



3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

[4] セグメンテーション

- セグメント単位で、仮想アドレス空間上のセグメント（仮想セグメントまたは論理セグメント）と実アドレス空間上のセグメント（実セグメントまたは物理セグメント）とをマッピング
- 仮想アドレス空間には仮想セグメントごとに仮想セグメント番号を、実アドレス空間には実セグメントごとに実セグメント番号をそれぞれ振る
- 仮想セグメントと実セグメントのマッピングはセグメントテーブルと呼ぶアドレス変換テーブルに記述
  - 仮想セグメントから実セグメントへのアドレス変換はセグメントテーブルを引く操作となる

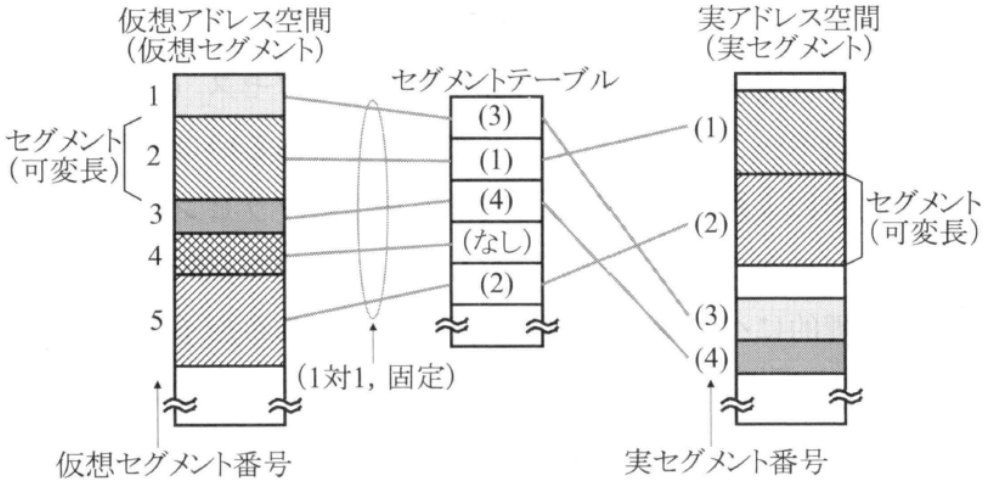
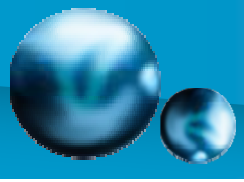


図 3.15 セグメンテーションによるマッピング（例）



3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

[4] セグメンテーション

- 長所

(1) セグメントは「論理的な意味」をもつ単位であるので、セグメントテーブルに記述しておく「セグメントの属性」をアクセス制御時に使用できる。また、実行時における動的な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になる

- セグメントの属性の例

(a)アクセス権
(b)命令かデータかの種別
(c)データ型;
(d)他のセグメントとの関係
(e)共用か非共用かの区別

(2) セグメント単位でのマッピングとなるので、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発生しない。もちろん、外部フラグメンテーションは発生する。

[4] セグメンテーション

- 短所

(3)セグメントサイズは可変かつ自由

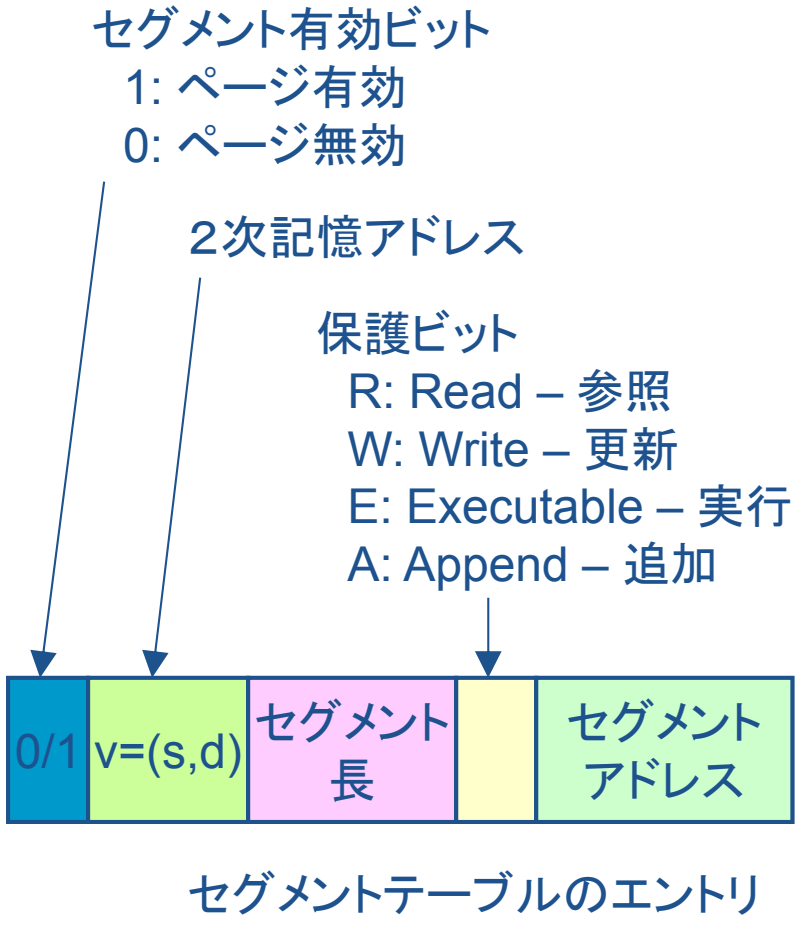
- メインメモリ（実メモリ）やファイル装置（バックアップメモリ）での領域管理、および、実メモリと仮想メモリとのマッピングの管理は非常に複雑
- メインメモリ（実メモリ）での外部フラグメンテーションが発生しやすい

→それを過ぎると、メインメモリの利用効率は格段に悪くなる

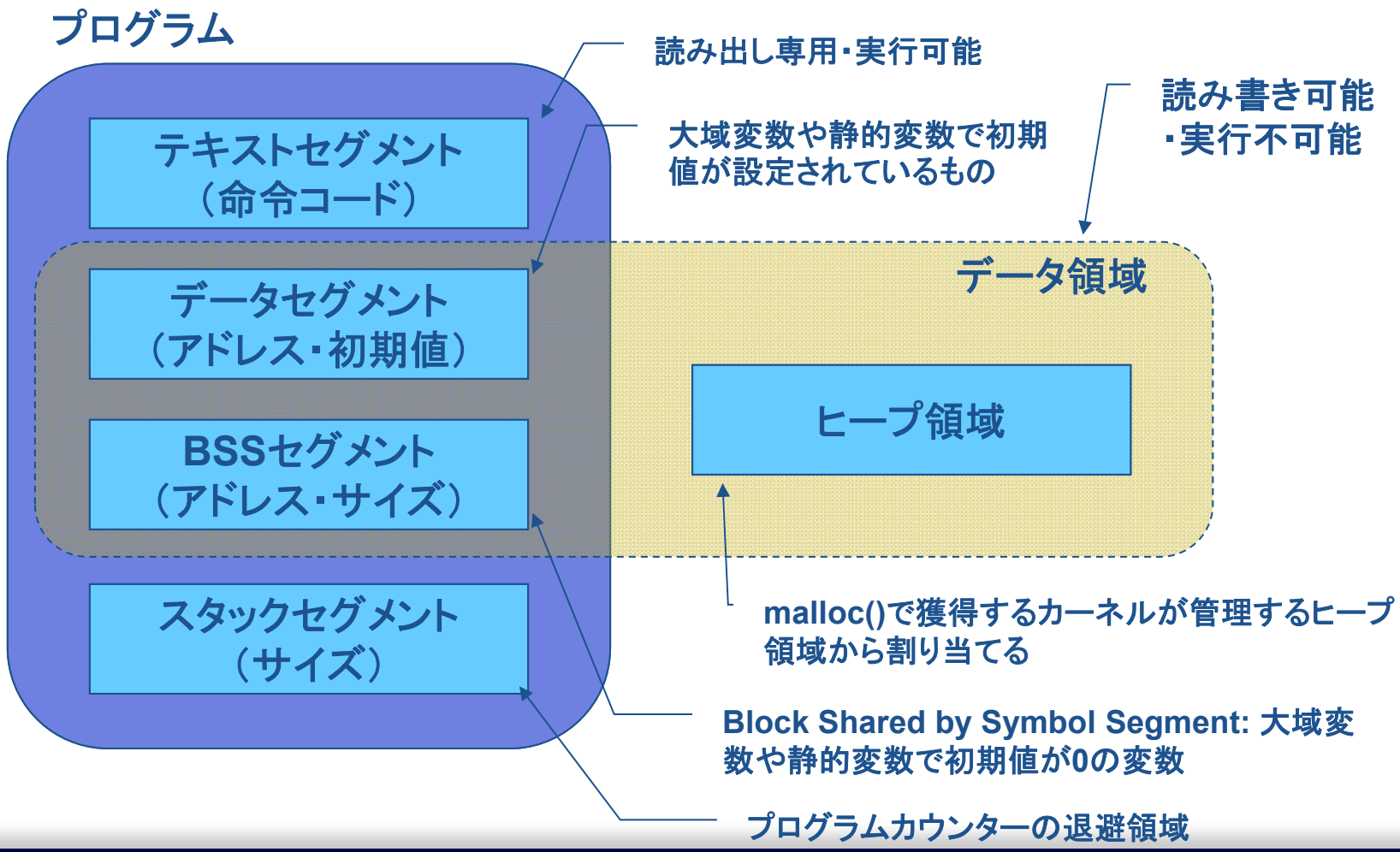
(4)メインメモリサイズより大きな仮想アドレス空間をセグメントとしてマツピングできないので、その場合に長所の(1)(2)を喪失してしまう

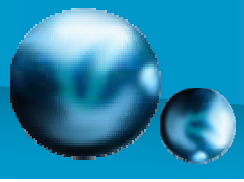
セグメンテーションの例

- プログラムは本来、論理的な単位（セグメント）の集合
  - 手続き、関数、配列
  - ユーザは名前で参照する
- セグメンテーション方式
  - プログラムのアドレス空間をセグメントに分割する
  - セグメント内の要素
    - セグメント先頭からのオフセットで識別
    - 論理アドレス＝＜セグメント、オフセット＞
  - 詳細なアクセス制御が可能
    - アクセス制御：セグメントの参照、更新、実行、追加など
    - 1つのセグメントは同じ保護モードでよい
    - アクセス制御の記述データ量が少しですむ



C言語のセグメント


プログラム
テキストセグメント
（命令コード）
データセグメント
（アドレス・初期値）
BSSセグメント
（アドレス・サイズ）
スタックセグメント
（サイズ）
読み出し専用・実行可能
大域変数や静的変数で初期
値が設定されているもの
読み書き可能
・実行不可能
データ領域
ヒープ領域
malloc()で獲得するカーネルが管理するヒープ
領域から割り当てる
Block Shared by Symbol Segment: 大域変
数や静的変数で初期値が0の変数
プログラムカウンターの退避領域



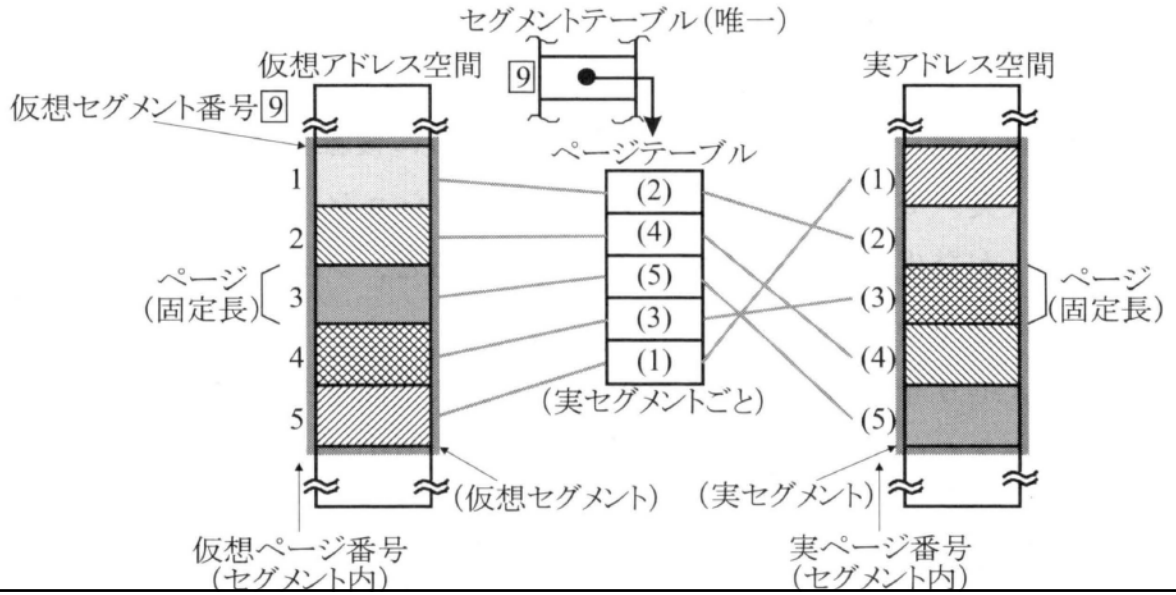
演習問題3.3

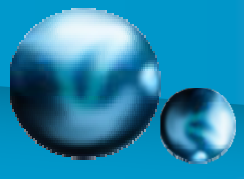
- 仮想メモリ方式にページング、セグメンテーションを採用した時、
  - 外部フラグメンテーションは起こるか？
  - 内部フラグメンテーションは起こるか？

3.2 仮想メモリーメインメモリの隠ぺい一
3.2.3 マッピング

[5] ページセグメンテーション

- 2種類のマッピングを行う
  1. 全体では、セグメント単位(セグメンテーション)
  2. 各セグメント内では、ページ単位(ページング)
- 仮想アドレス空間も実アドレス空間も一定かつ固定長のページサイズで分割する
  1. 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する
     セグメントはひとまとまりであるので、その全体サイズは可変かつ自由
     仮想セグメントと実セグメントのマッピングはセグメントテープルに記述してある
  2. ページ単位で、仮想セグメントと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(仮想ページと実ページ)をマッピングする
     仮想ページと実ページのマッピングは各実セグメントごとに備えるページテーブルに記述してある





3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

[5] ページセグメンテーション（続き）

- 仮想セグメントをページ単位で分割して、それら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におけるアドレス変換の方法
  1. セグメントテーブルを引くことによって、仮想セグメント番号から実セグメントごとに備えるページテーブルのアドレスを得る
  2. 1で得たページテーブルを引くことによって、仮想ページ番号から実ページ番号を得る
- OSによるメモリ管理の観点から見たページセグメンテーションの特徴

  長所

  (1)ページングとセグメンテーシヨンの両方式の長所を融合してもつ

  最初のマッピングは論理的なまとまりのセグメント単位で行うので、そのセグメントの「論理的意味」を活用できる

  セグメントのマッピングを決定後は通常のページングによるので、実メモリの管理が簡単で、利用効率も優れている

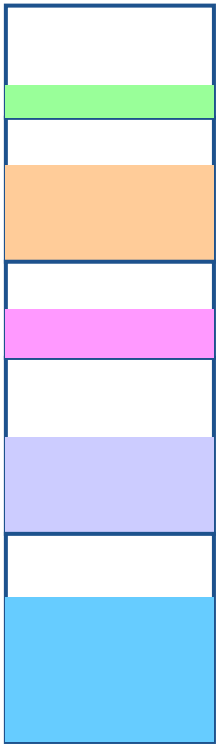
  短所

  (2) 1個のセグメントテーブルと実セグメント総数分のページテーブル（どれもすべてが可変長）の2種類のアドレス変換テーブルが必要となる

  テーブル全体が占める領域（空間）、および、2段階のアドレス変換すなわちテーブル検索にかかる時間、のそれぞれが大きくなる

実際には

ページング方式



- Pentiumの例
  - セグメンテーション方式の利点（サイズの変動するテーブルを利用できる、アクセス保護ができる）を生かしたページング

ページセグメンテーション方式

3.2 仮想メモリーメインメモリの隠ぺいー
3.2.3 マッピング

[6] [まとめ]仮想メモリへのプロセス割り付けの実際

- ユーザプロセス領域ごとに、それを1個の独立した仮想アドレス空間とする
  - 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とする
  - 実メモリ(メインメモリ)へマッピングする、すなわち割り付けるユーザプロセス領域(各々セグメント)のために、バックアップメモリ(ファイル装置)上に、スワップ領域という一定サイズの領域をあらかじめ確保しておく
  - OS自身は、仮想メモリ(ブロック置換)の対象外とし、実メモリ(メインメモリ)に一定サイズの領域をOS自身で直接確保し、その領域に常駐する
- 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る
  - 仮想メモリ機構が稼働している場合、この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、必要とするプロセスをファイル装置から読み出してメインメモリに置く(割り付ける)
  - 不要と判定できる(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

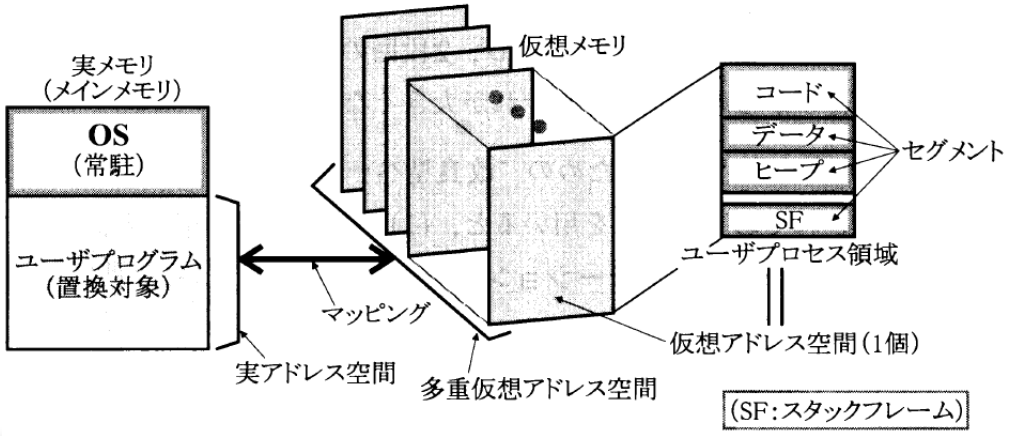
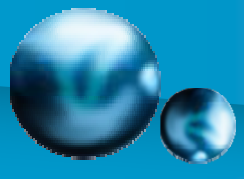


図 3.17 仮想メモリへのプロセス割り付け



3.2 仮想メモリーメインメモリの隠ぺいー
3.2.4 置き換え

[1]【まとめ】ページフォールトとブロック置換（再掲）

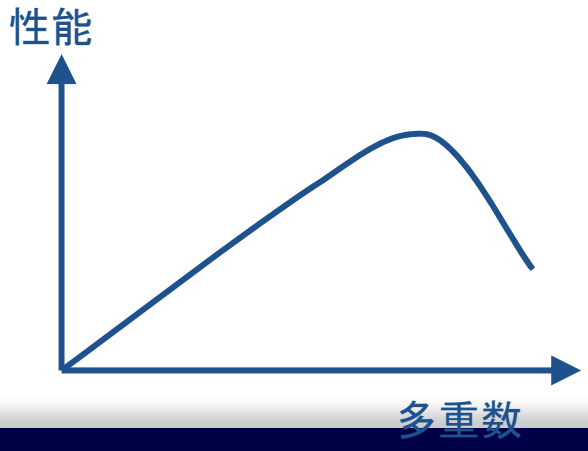
- ページングやページセグメンテーションというページを単位とするマッピング
  - アクセスを要求した仮想ページがメインメモリ（実メモリ）上にない、すなわちページフォールトである場合に生じる割り込みがページフォールト割り込み
  - ページフォールト割り込みは、「アドレス変換時に、ページテーブルにアクセス対象の仮想ページの登録がない、すなわち、ページフォールトが生じる」場合に発生
- ページフォールト割り込みによって、ページフォールトという通知を受けたOSは、その割り込み処理において、以下のブロック置換（ページングやページセグメンテーションの場合はページ置換）を実行する
  1. ファイル装置（仮想メモリのバックアップメモリ）に格納してある当該仮想ページをメインメモリ（実メモリ）の不要な実ページと置換（スワップ）する
  2. ページテーブルを書き換える
- ページ置換はページ単位で実行

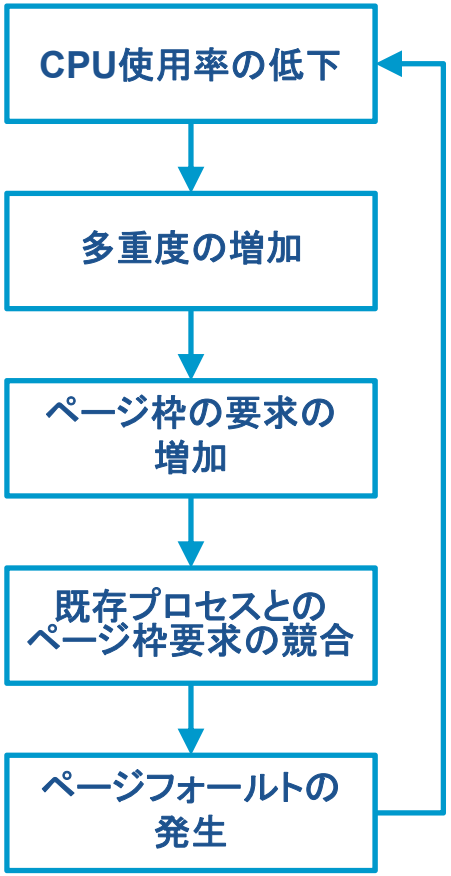
  (A) ページアウト (page-out)：メインメモリからのスワップアウト

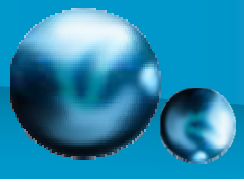
  (B) ページイン (page-in)：メインメモリへのスワップイン

スラッシング

- スラッシング
  - 仮想メモリ機構では、実メモリ容量が小さすぎると、実メモリ上での参照局所性の利用が不可能となり、ブロック置換が頻発する
  - プロセッサがOS機能であるブロック置換管理にかかりきりとなって、ユーザプロセスおよびブロック置換管理以外のOS機能の実行が実質上不可能となること
  - スラッシングはオーバヘッドであるので、仮想メモリの効果を生かすためには、充分な容量の実メモリを装備かつ確保して、スラッシングの発生を防止する必要がある
- マルチプログラミング環境において右のような悪循環に陥る







3.2 仮想メモリーメインメモリの隠ぺい―
3.2.4 置き換え

[2] ページ置換アルゴリズム

- 使用頻度すなわち参照局所性が高いページをメインメモリ（実メモリ）で保持する

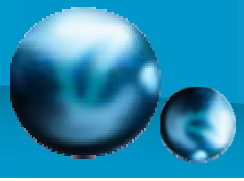
  →使用頻度、すなわち参照局所性が低い実ページをスワップアウトすることが有効となる

  →ページ置換アルゴリズムの目標は、参照局所性が低いページを決定すること
- ページフォールトが発生すると、OSはまずページ置換アルゴリズムによって、スワップアウトするページを決定

  →ページ置換アルゴリズムが不適切であると、適度なサイズの実メモリ領域が確保できていてもスラッシングを引き起こしてしまう
- ページ置換アルゴリズムの評価指標

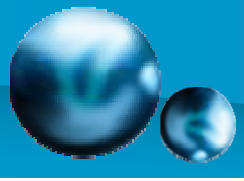
  (a) ページのライフタイム：ページフォールト間の平均時間間隔すなわちページの平均（実メモリ）滞在時間

  (b) ページフォルト率：ライフタイムの逆数



演習問題3.4

- 以下を仮定する
  - ページフォールトの処理に20ms要するとする
  - ページフォールト以外ではブロックしないCPUバウンドのプログラムを考える
  - プロセス起動直後の過渡状態は無視する
- ページフォールト率が5回／秒と0回／秒ではスループットはどのように異なるか？



3.2 仮想メモリーメインメモリの隠ぺい―
3.2.4 置き換え

[3] 代表的なページ置換アルゴリズム

1. LRU (Least Recently Used)
   - 最後のアクセス時刻が最古であるページを最優先のスワップアウト対象とする
   - 長所：時間的参照局所性を自然に反映しているので、ページフォールト率は低くなる
   - 短所：アクセス履歴を記録し比較するページ置換管理機構は複雑で、ページ置換時間は長くなる
2. FIFO (First In First Out)
   - メインメモリに一番最初にすなわち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とする
   - ページ置換機構の実装コストはLRUとランダムとの中間
   - アクセスや参照はチェックしないので、参照局所性の反映度はLRUよりも低い
3. ランダム (random)
   - 任意あるいは無作為にスワップアウトするページを決定する
   - 長所：簡単に実現あるいは実装できる
   - 短所：参照局所性をまったく考慮していない、すなわち、アルゴリズム（戦略）とはいえない
4. ワーキングセット (working-set)
   - ワーキングセットではないページを優先すべきスワップアウト対象とする
   - ワーキングセットではないページのうちから、他のアルゴリズムによってスワップアウト対象を決定する

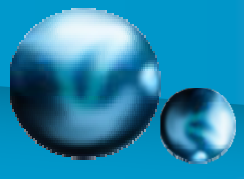
(1)～(4) の外にも種々のページ置換アルゴリズムがある

- 実際には、「ページ置換アルゴリズムよりも、アクセスや参照の対象であるプログラムやデータブロックそのものが示す参照局所性がページフォールト率を左右する
- どのアクセスや参照の対象（プログラムやデータ）に対してもページフォールト率が低くなるような最適なページ置換アルゴリズムの設計や選定は困難

ページ置換えアルゴリズム：FIFO

- メインメモリ上の最も古いページを置き換える
- 実装
  - メモリ上のすべてのページに対するFIFOキュー
  - 置換えが必要になると、キュー先頭のページを選び、ページアウト
  - ページイン⇒キューの最後につなぐ

ページフォールト	p	p	p	p	p	p	p			p	p	
参照ストリング	0	1	2	3	0	1	4	0	1	2	3	4
ページ枠の内容（FIFOキュー）	0	0	0	1	2	3	0	0	0	1	4	4
		1	1	2	3	0	1	1	1	4	2	2
			2	3	0	1	4	4	4	2	3	3



演習問題3.5

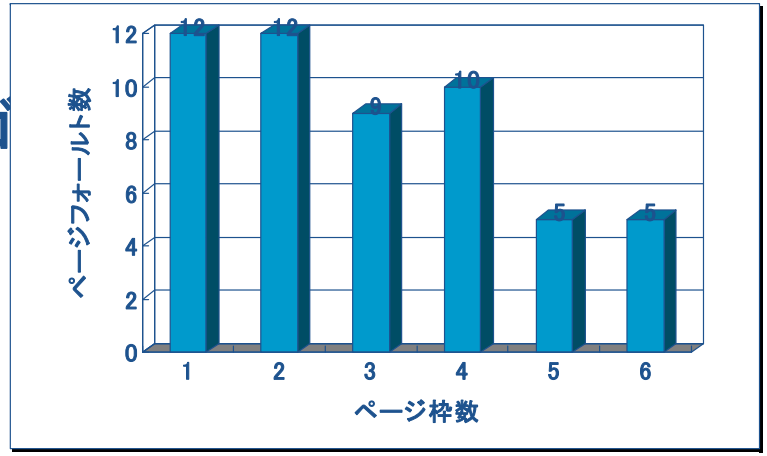
- 置き換えアルゴリズムとしてFIFOを考え、参照列が０１２３０１４０１２３４のとき、ページ枠数が１、２、４、５の場合はどうなるか？ページフォールト数を求め、ページ枠数とページフォールト回数の対応表を作れ

Beladyの異常（Belady’s nomaly）

- メモリを増強してページ枠を4に増やすとフォールト率がかえって増加する
  - FIFO異常

ページフォールト				p	p	p	p			p	p	p	p	p	p
参照ストリング	0	1	2	3	0	1	4	0	1	2	3	4			
ページ枠の内容（FIFOキュー）	0	0	0	0	0	0	1	2	3	4	0	1			
		1	1	1	1	1	2	3	4	0	1	2			
			2	2	2	2	3	4	0	1	2	3			
				3	3	3	4	0	1	2	3	4			

- Beladyの異常を起こさないアルゴリズム
  - 「スタックアルゴリズム」と呼ばれるクラス
  - OPT、LRUなど



ページ置換えアルゴリズム：OPT

- 別名MIN：最小のページフォルト率
- 選択するページ
  - 次回に参照されるのが最も遠い未来である
  - もちろん実用的ではない
  - 他のアルゴリズムの性能を評価するための比較対象

ページフォールト	p	p	p	p			p			p	p	
参照ストリング	0	1	2	3	0	1	4	0	1	2	3	4
ページ枠の内容	0	1	2	3	3	3	4	4	4	4	4	4
		0	1	1	1	1	1	1	1	2	3	3
			0	0	0	0	0	0	0	0	0	0

ページ置換えアルゴリズム：LRU

- 主記憶にあるページのうち最も長い間参照されていないページを選択
  - 実際に利用される例が多く、よいと考えられている
- ただし、ハードウェアによる支援が必要
  - スタック：参照されたページ番号をスタックの最上部に入れる。スタックの最下部がもっとも古いものになり、そのページを選択する
  - カウンタ：メモリ参照のたびに +1、OSは一定時間間隔ごとにリセット（厳密にはLeast Frequently Used）
- あるいは参照ビットにより、近似的にLRUを実行
  - プログラム実行時に、参照されたページの参照ビットを立てる
  - 定期的にOSが参照ビットを0にする
  - 順番はわからないが、参照されていないページはわかる

ページフォールト	p	p	p	p	p	p	p			p	p	p
参照ストリング	0	1	2	3	0	1	4	0	1	2	3	4
ページ枠の内容（スタック）	0	1	2	3	0	1	4	0	1	2	3	4
		0	1	2	3	0	1	4	0	1	2	3
			0	1	2	3	0	1	4	0	1	2

スタックアルゴリズム

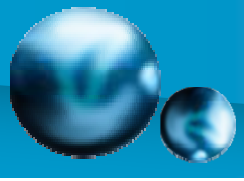
- スタックアルゴリズム
  - 参照ストリングの各要素において、[ページ枠*p*個の場合メインメモリに置かれるページ集合]⊂[*p*+1個の場合のページ集合]
  - 右はLRUで3ページ枠、4ページ枠の例
- メインメモリを増やすとページフォールト率は下がる。問題は、良いとされているLRUがFIFOより悪いこと(3ページ枠のとき)
- どの場合にも通用する最適解はない
  - ある与えられた資源量に対して最適なアルゴリズムはある
  - 資源量を変化させると最適なアルゴリズムは変わる
  - どんな資源量に対しても最適なものはないが、「だいたいよい」ものはある
  - ただし、資源量が限定されている場合(例:PDA)は詳細な検討が必要
  - アーキテクチャデザインの重要性

ページフォールト	p	p	p	p	p	p	p			p	p	p
参照ストリング	0	1	2	3	0	1	4	0	1	2	3	4
ページ枠の内容(スタック)	0	1	2	3	0	1	4	0	1	2	3	4
		0	1	2	3	0	1	4	0	1	2	3
			0	1	2	3	0	1	4	0	1	2

ページフォールト	p	p	p	p			p			p	p	p
参照ストリング	0	1	2	3	0	1	4	0	1	2	3	4
ページ枠の内容(スタック)	0	1	2	3	0	1	4	0	1	2	3	4
		0	1	2	3	0	1	4	0	1	2	3
			0	1	2	3	0	1	4	0	1	2
				0	1	2	3	3	3	4	0	1

ページ置換アルゴリズムの評価方法

- 置換えアルゴリズムの評価方法
  - 仮想アドレス空間の特定のアドレス参照列（参照ストリング）を用いてアルゴリズムを評価
  - 参照ストリングの生成
    - 乱数発生器
    - 既存のプログラムの実行時のトレースデータ
- 評価プログラム
  - 入力
    - 参照ストリング
    - ページ枠数
  - 出力
    - 置換え要・不要の列、置換えるページ番号の列⇒ページフォールト率



演習問題3.6

- 以下のプログラムを実行した時の参照ストリングを求めよ。

```
int s[64], d[64];
register int i;
for (i=0; i<64; i++) d[i] = s[i];
```

- ただし、プログラムはページ0、配列sはページ1、配列dはページ2に割り当てられるものとせよ。
- 同じページへの繰り返し参照は1つにまとめて書いてよい
  - 11100221⇒1021

参照局所性の実際（再掲）

- type 1: write（赤）
- type 2: read（青）
- type 3: execute（緑）

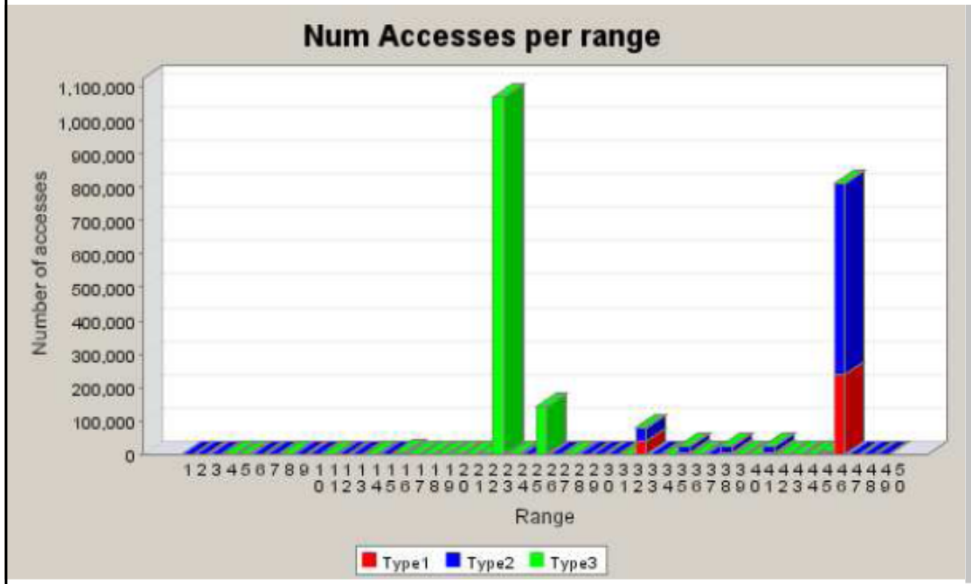


Fig. 2. Access pattern of m-fft

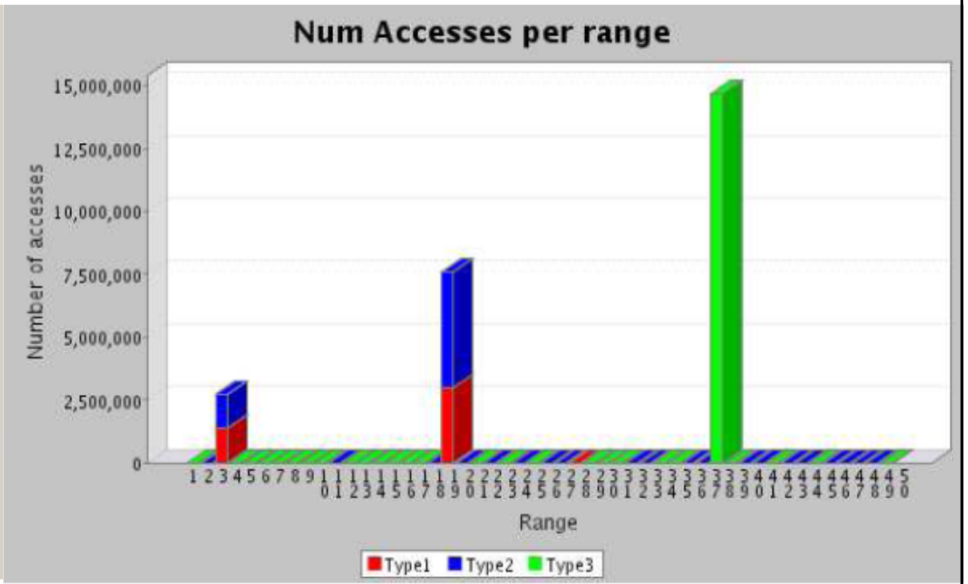
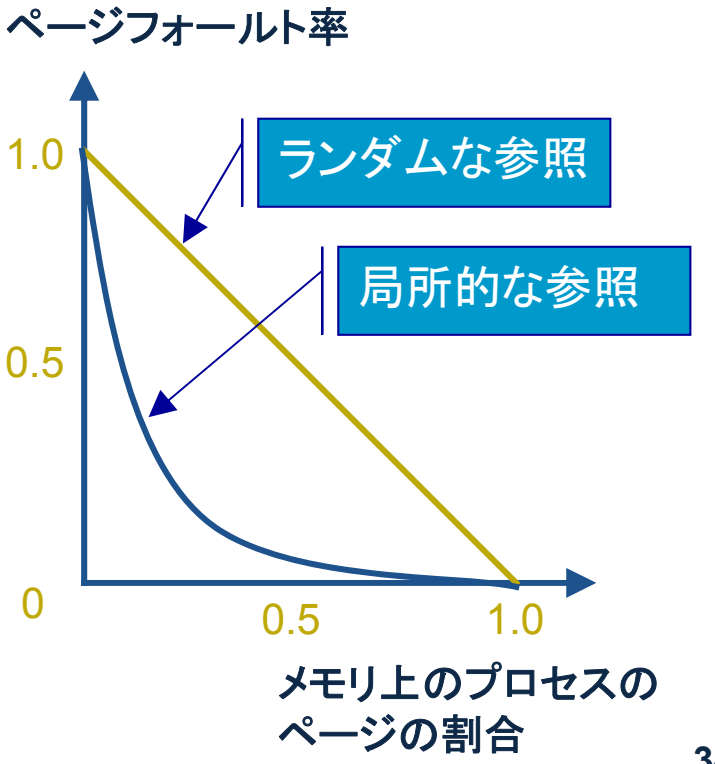


Fig. 1. Access pattern of tower of Hanoi program

33

参照局所性の効果

- 参照局所性
  - 各プロセスが参照するアドレスには偏りがあり、特定の部分にアクセスが集中（後の例参照）：空間的局所性
    - プログラムの10%がアクセス回数の90%を占める
  - 「よくアクセスするページ集合」は時間を追って変化するが、短期的には余り変わらない：時間的局所性
- メモリ量／プロセスサイズ≧ 0.3～0.8
  - ページフォルト率はほぼ0になるといわれている



ページ置換えアルゴリズムの実際

- A. Aggrawal, “Page Replacement Algorithm Simulator”
  - プログラムの実行時のトレースデータを使用
- LRUはよいが、その近似アルゴリズムCLKはさほどでもない（左図）
- いつも理屈どおりとは限らない（右図）

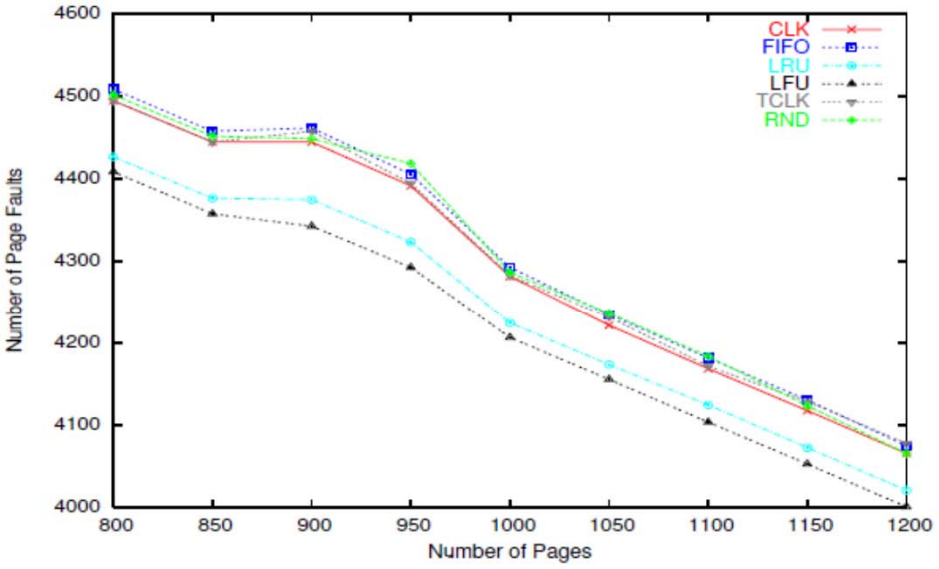


Fig. 9. Performance of page replacement algorithms on m-sor program

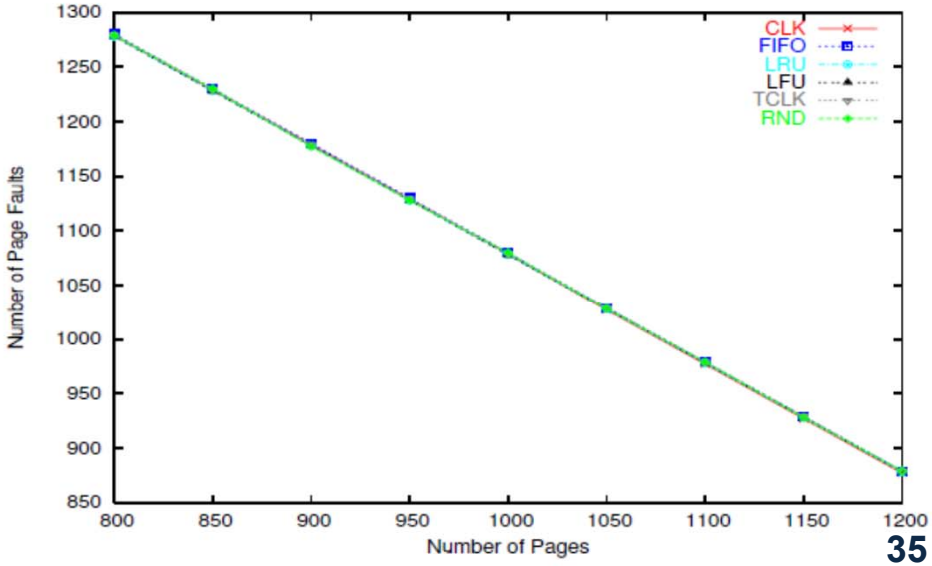


Fig. 13. Performance of page replacement algorithms on sacc program

ワーキングセットモデル

- Denningが提案したプロセスの振る舞いのモデル
- ワーキングセット：プロセスの実行の各時点で、プロセスが活発に参照するページの集合
  - ワーキングセットがメインメモリに在ればよい
  - ない場合はページフォルトが頻発する：スラッシング
- ワーキングセット *W*(*t*, *w*) := [*t*-*w*, *t*]の間にプロセスによって参照されるページ集合
  - ここでの時間は仮想時間→他のプロセスが実行される時間を含めない
  - 変数*w*: ウィンドウサイズ
- ワーキングセット法：ワーキングセットをメモリにできるだけ保持する技法→マルチプログラミング環境で有用になる

w=5の場合

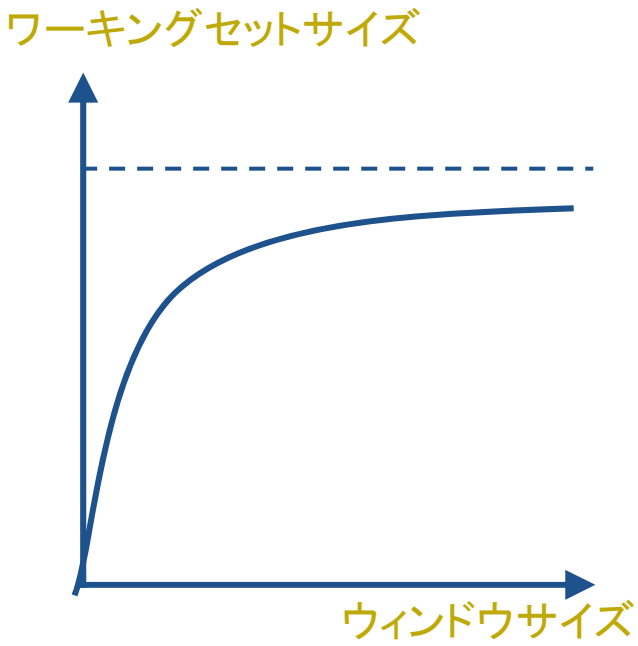
ページフォールト	p	p	p	p	p	p	p			p	p	p
参照ストリング	0	1	2	3	0	1	4	0	1	2	3	4
ページ枠の内容	0	1	2	3	0	1	4	0	1	2	3	4
		0	1	2	3	0	1	4	0	1	2	3
			0	1	2	3	0	1	4	0	1	2

36

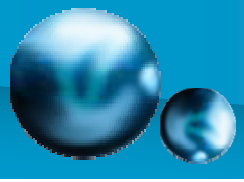
ワーキングセット法の問題

- ワーキングセットのウィンドウサイズ (*w*) が効率に大きな影響を与える
  - *w*がちいさすぎる
    - ワーキングセット全体を取り込めない
  - *w*が大きすぎる
    - 複数のワーキングセットを取りこんでしまう
    - 無限大にすると、プログラム全体となってしまう(なぜだめなのか？)
- 各プロセスのワーキングセットのサイズ
  - 大きさの総和＞全ページ枠数になると、ページ枠不足

    →スラッシングの発生
  - ワーキングセットは変化する
    - ある時点のワーキングセットサイズは将来のワーキングセットサイズと同じではない



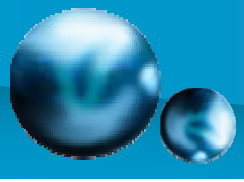
ワーキングセットサイズ曲線



3.2 仮想メモリーメインメモリの隠ぺい
3.2.4 置き換え

[4] ページ置換のタイミング

- 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）か？
  - (A) デマンドページング（要求時ページング）:
    - プログラム（実行中のプロセス）自身が実行時（動的）にアクセスや参照を要求（デマンド）するページをそのたびに読み込む。OS機能をハードウェア機構が支援する
    - OSとハードウェアの機能分担する動的ページング
    - 長所
      - (1) 必要なすなわち要求するページだけを読み込む点では無駄がない
    - 短所
      - (2) プロセス実行の開始時、すなわち、初めての実行中状態への選移時にページフォールトが集中発生するので、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かつ空間的な冗長性が存在する
    - 適切なページ置換アルゴリズムの選択が重要な要件となる



3.2 仮想メモリーメインメモリの隠ぺいー
3.2.4 置き換え

[4] ページ置換のタイミング（続き）

(B)プリページング（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（通常は複数個）を実行前（静的）にまとめて読み込む
- OS機能をコンパイラ機能が支援する
- OSとコンパイラの機能分担による静的ページング
- 長所

  (1) 予測の的中率が高ければ、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り、高速化が達成できる;
- 短所

  (2) 不要なページも読み込んでしまう可能性がある

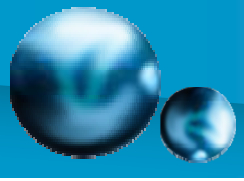
  (3) 予測の的中率が低いと、予測に要するオーバヘッドが顕在化する。予測に要するオーバヘッドを上回る（ページ置換そのものの）高速化が必須の要件となる

- 現代では、メインメモリの実装コストが低くなって、大容量のメインメモリ（実メモリ）を実装できるのが普通

  →大容量のメインメモリを実装していれば、余分で冗長なページも読み込んでおくことができる

  → 予測がはずれる、すなわち「読み込んだページを使用しない」場合の影響は少ない

  → (B)のプリページングを採用するOSが多い

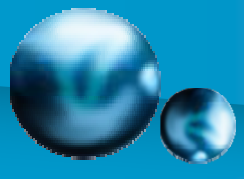


3.2 仮想メモリーメインメモリの隠ぺいー
3.2.5 メモリ保護

[1] 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、アクセス権にしたがって、相互に保護し合う
- OSのメモリ保護機能はOSがアクセス権を管理することによって実現
- 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があれば(メモリ保護違反)、それを要因とする割り込みが発生する
- OSは、メモリ保護違反による割り込みを受け付けると、ユーザプログラムの実行(プロセッサ状態はユーザ状態)からOSの実行(プロセッサ状態はカーネル状態)に切り替えて、メモリ保護違反に対する割り込み処理を行う

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をアクセス権という
- アクセス形態の例:読み出し、書き込み、(命令としての)実行など。
- アクセス権はプロセス(プログラム)ごとに設定



3.2 仮想メモリーメインメモリの隠ぺい一
3.2.5 メモリ保護

[2] OSによるメモリ保護一例:仮想メモリ機構への組み込み一

- ページングやページセグメンテーション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する
  - (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する
- (1) (2)ともに仮想メモリ機構の一部として実現可能
- 特に、多重仮想アドレス空間に(2)を適用すれば、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し運用できる