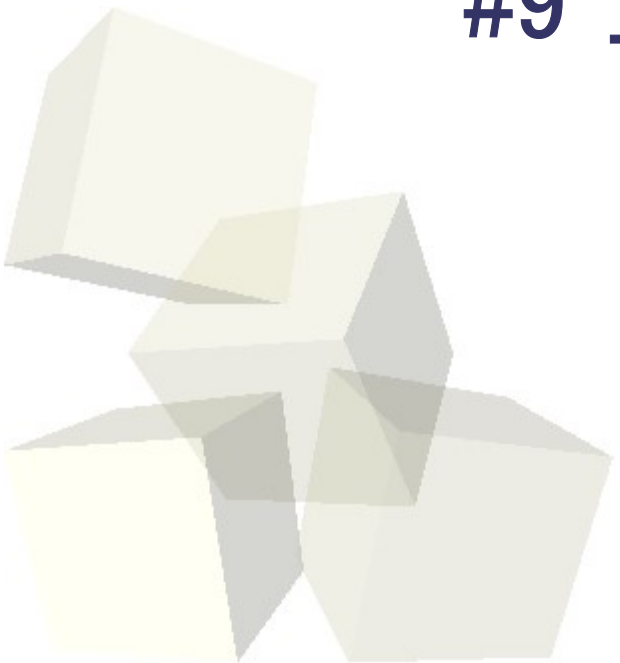
オペレーティングシステム

#9 主記憶管理：ページング



復習

主記憶管理

- プログラムの実行には，ディスクから主記憶への
  プログラムのロードが必要
- マルチプログラミング環境では複数プロセスが
  並行して主記憶を使用
- 主記憶上でプログラムおよびデータを格納するための
  領域を管理する必要

記憶領域確保の方式

- 固定区画方式
- 可変区画方式

復習

- 可変区画方式
  - 空き領域の割り当て方式
    - → ベストフィット
    - → ワーストフィット
    - → ファーストフィット
  - 割当を適切に行わないと，プロセスが使用できない断片的な空き領域が多く発生
    （フラグメンテーション）
  - 空き領域の管理方式
    - → リスト方式
    - → ビットマップ方式

復習

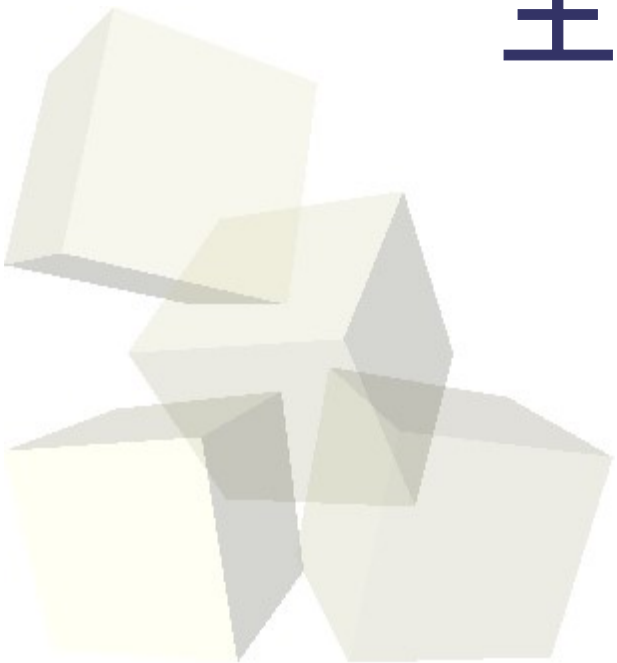
静的ライブラリ

- リンク時にロードモジュールに埋込み（静的リンク）
- 複数プログラムで使用されるライブラリがある場合，主記憶領域の無駄

共有ライブラリ

- ロードモジュールはリンク情報のみを持ち，実行時にリンク（動的リンク）
- 複数プログラムで使用される場合でも，各ライブラリは1つのイメージだけ主記憶上に存在すればよい
- 主記憶領域（およびディスク領域）の有効活用

9.1
主記憶の動的再配置



仮想化

- 有限のリソースを無限に見せる

- 空間分割
  - ハードウェアリソースを複数領域に区切る
  - CPUでは不可能
- 時間分割
  - 複数プログラムが交互に使う
  - CPU　　:タイムシェアリング
  - 主記憶　:オーバレイ

主記憶の時間分割による多重化

- 主記憶の動的再配置
  - オーバレイをより一般化した手法
- 主記憶に配置する対象
  - プログラム領域:
    現在実行中のプログラムカウンタの近傍
  - データ領域:
    最近アクセスした領域の近傍
  - それ以外は二次記憶装置(ハードディスク)に

オーバーレイ

```
main(){
        funcA();
}

funcA(){
        funcB();
}

funcB(){
        funcC();
}
        :
        :
```

（論理空間）

```
main:
        :
        ret;
funcA:
        :
        ret;
funcB:
        :
        ret;
funcC:
        :
        ret;

        :
        :
```

（物理空間）空き領域

仮想記憶：キーワード

仮想記憶

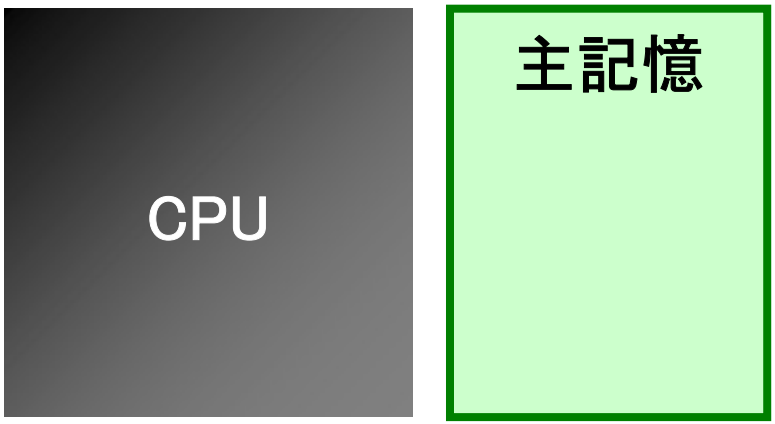
- 主記憶の動的再配置により，プロセスが使用できる主記憶領域を無限大にする
  - → 実際には，アドレスレジスタの有効ビット数を超えるアドレスにはアクセスできないし，ハードディスク容量にも依存するため，厳密には無限大ではない

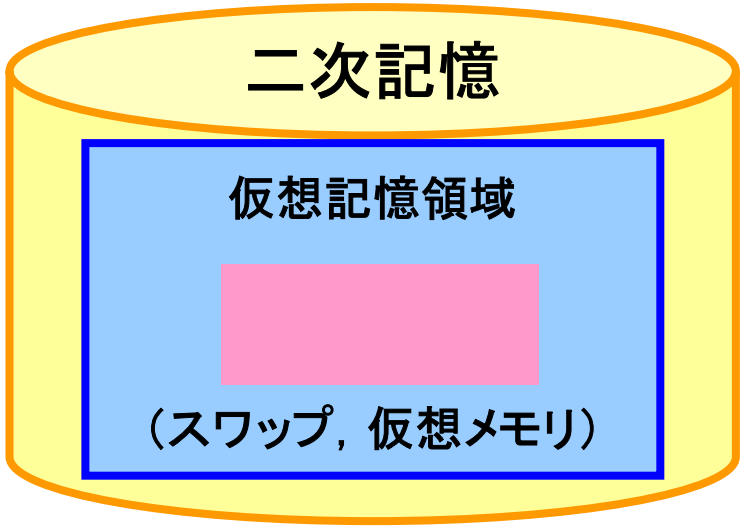
仮想アドレス

- 記憶容量に制限のない論理アドレス

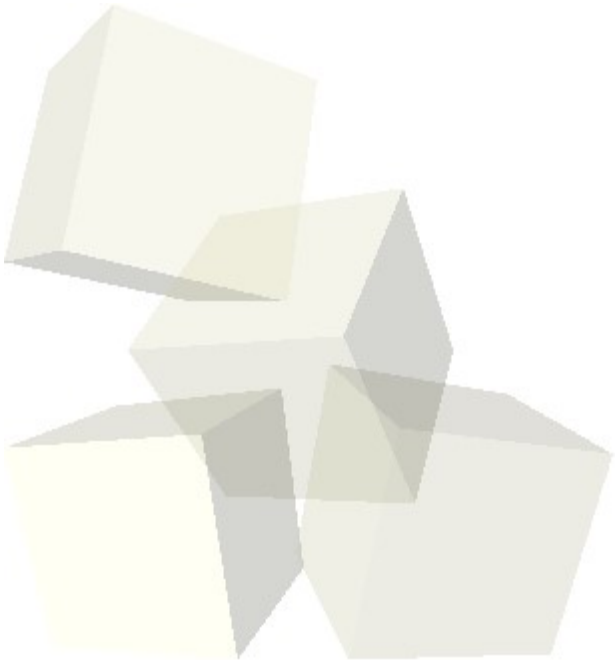
仮想記憶実現のための操作

- スワップイン
  - 実行中のプログラムが，必要となる領域を二次記憶から主記憶に転送する操作
- スワップアウト
  - 上記スワップインを行う際，その空き領域を確保するために，当面必要としない領域を主記憶から二次記憶に転送する操作





9.2
ページング



ページング

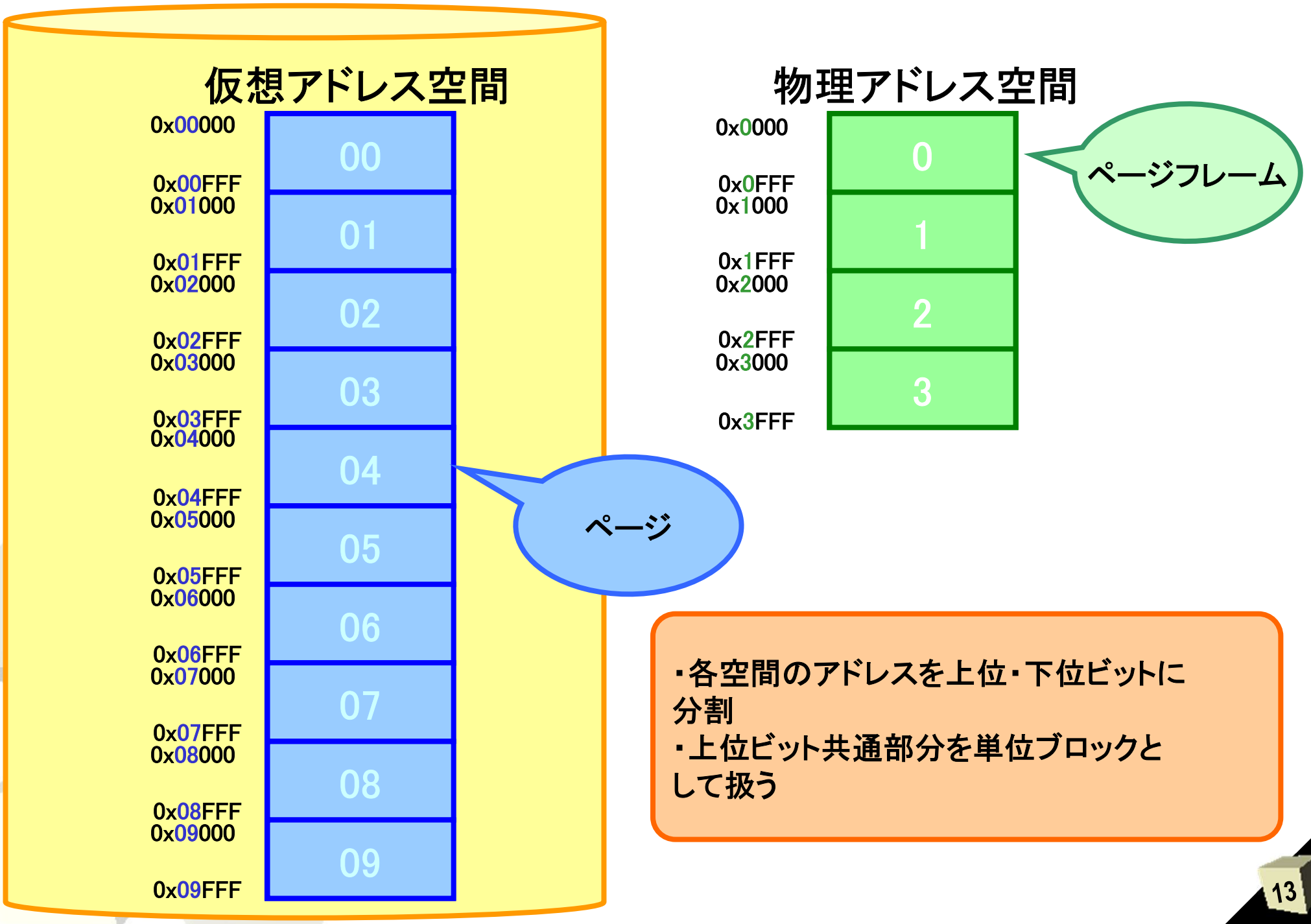
記憶領域の仮想化

- 仮想アドレス空間の一部が物理メモリに存在
- 仮想アドレスと物理アドレスの対応付けが必要
- 物理メモリ上に存在する「仮想記憶の一部」は時々刻々変化する（動的再配置）
- 対応付けも時々刻々変化

ページング

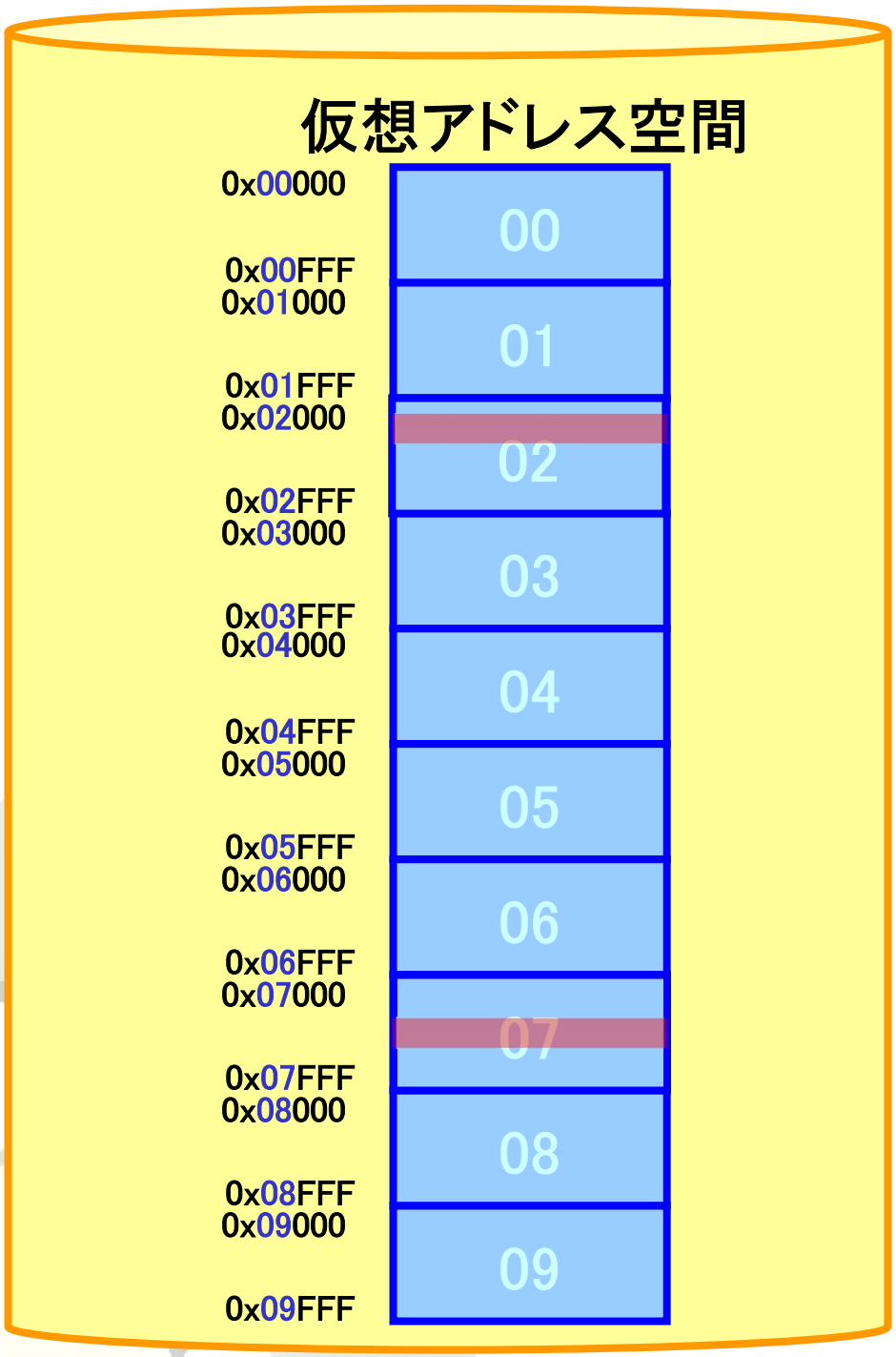
- 「ページ」と呼ばれる単位で動的再配置を行う

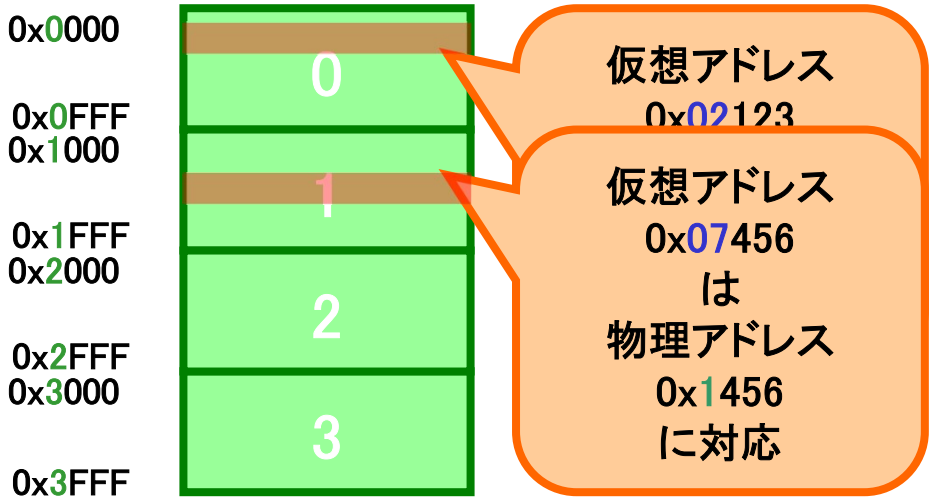
ページング



・各空間のアドレスを上位・下位ビットに分割
・上位ビット共通部分を単位ブロックとして扱う

ページング





下位ビットは共通
上位ビットの変換が必要

仮想	物理
02	0
07	1
:	:

ページテーブル

- 仮想アドレスから物理アドレスへの変換
- ページ番号からページフレーム番号への変換

ページ	V	P	C	ページフレーム
00				3
				-
0				0

Virtual Memory flag

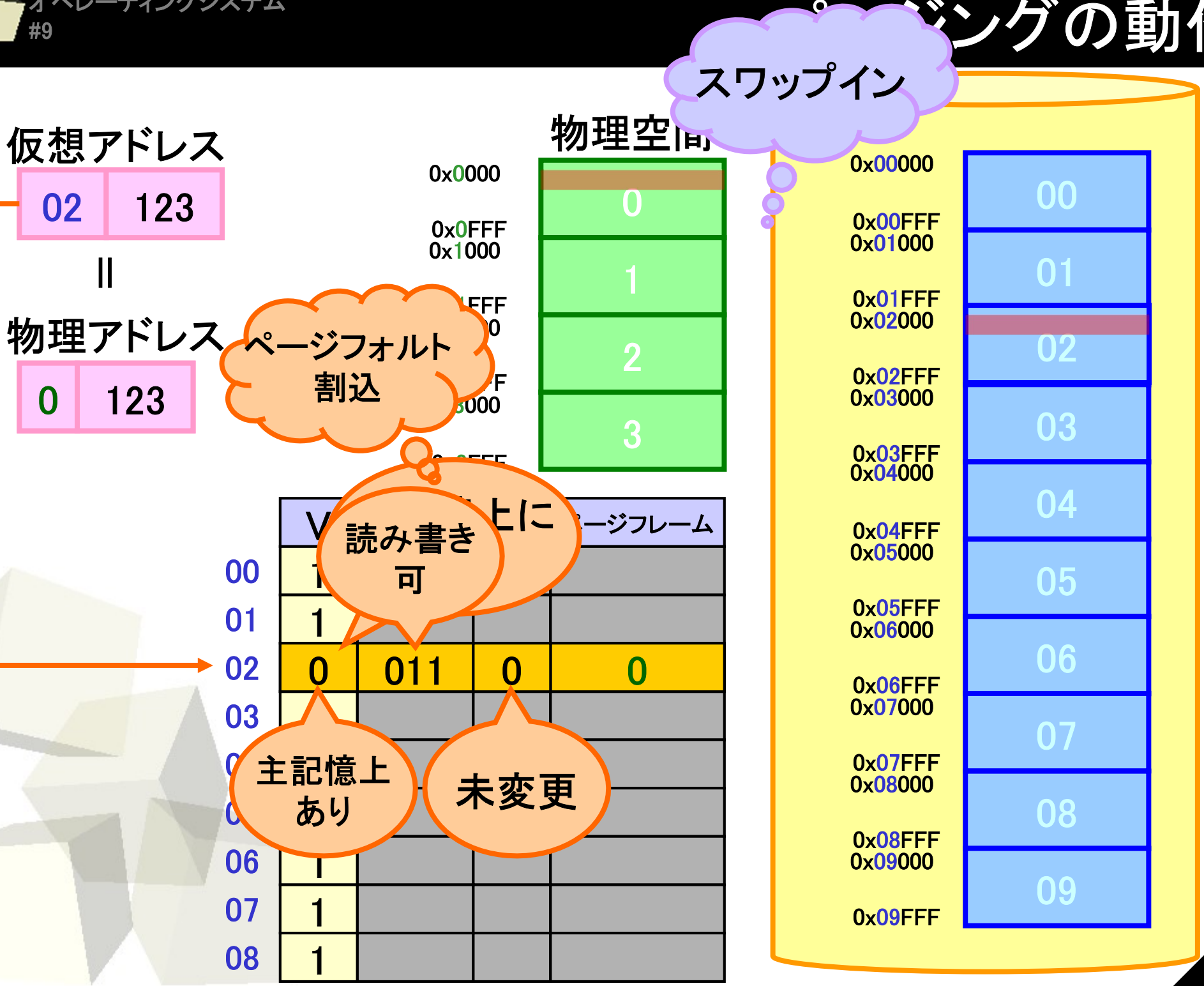
Permission flag

Change flag

ページテーブル

- Vフラグ (Virtual Memory flag)
  - そのページが主記憶上に存在するか否かを示す
- Pフラグ (Permission flag)
  - そのページに対するアクセス条件を表す
  - 例）読み込み可，書き込み可，など
- Cフラグ (Change flag)
  - スワップイン後，そのフレームに対して書き込みが行われたか（変更されたか）否かを示す

ページングの動作


スワップイン
仮想アドレス
02
123
=
物理アドレス
0
123
物理空間
0x0000
0x0FFF
0x1000
0
1
2
3
ページフォルト
割込
読み書き
可
主記憶上
あり
未変更
00
01
02
03
04
05
06
07
08
0
1
1
0
011
0
0
1
1
1
0x00000
0x00FFF
0x01000
0x01FFF
0x02000
0x02FFF
0x03000
0x03FFF
0x04000
0x04FFF
0x05000
0x05FFF
0x06000
0x06FFF
0x07000
0x07FFF
0x08000
0x08FFF
0x09000
0x09FFF
00
01
02
03
04
05
06
07
08
09

ページングの動作

仮想アドレス

07	456

||

物理アドレス

1	456

物理空間

0x0000
0x0FFF
0x1000
0x1FFF
0x2000
0x2FFF
0x3000
0x3FFF

02
1
2
3

	V	P	C	ページフレーム
00	1			
01	1			
02	0	011	0	0
03	1			
04	1			
05	1			
06	1			
07	0	011	0	1
08	1			

主記憶上にない

0x00000
0x00FFF
0x01000
0x01FFF
0x02000
0x02FFF
0x03000
0x03FFF
0x04000
0x04FFF
0x05000
0x05FFF
0x06000
0x06FFF
0x07000
0x07FFF
0x08000
0x08FFF
0x09000
0x09FFF

00
01
02
03
04
05
06
07
08
09

ページングの動作

仮想アドレス

=

物理アドレス

物理空間

0x0000	02
0x0FFF 0x1000	07
0x1FFF 0x2000	05
0x2FFF 0x3000	00
0x3FFF	

	V	P	C	ページフレーム
00	0	011	0	3
01	1			
02	0	011	0	0
03	1			
04	1			
05	0	011	0	2
06	1			
07	0	011	0	1
08	1			

0x00000	00
0x00FFF 0x01000	01
0x01FFF 0x02000	02
0x02FFF 0x03000	03
0x03FFF 0x04000	04
0x04FFF 0x05000	05
0x05FFF 0x06000	06
0x06FFF 0x07000	07
0x07FFF 0x08000	08
0x08FFF 0x09000	09
0x09FFF	

ページングの動作

仮想アドレス

03	789

||

物理アドレス

0	789

物理空間

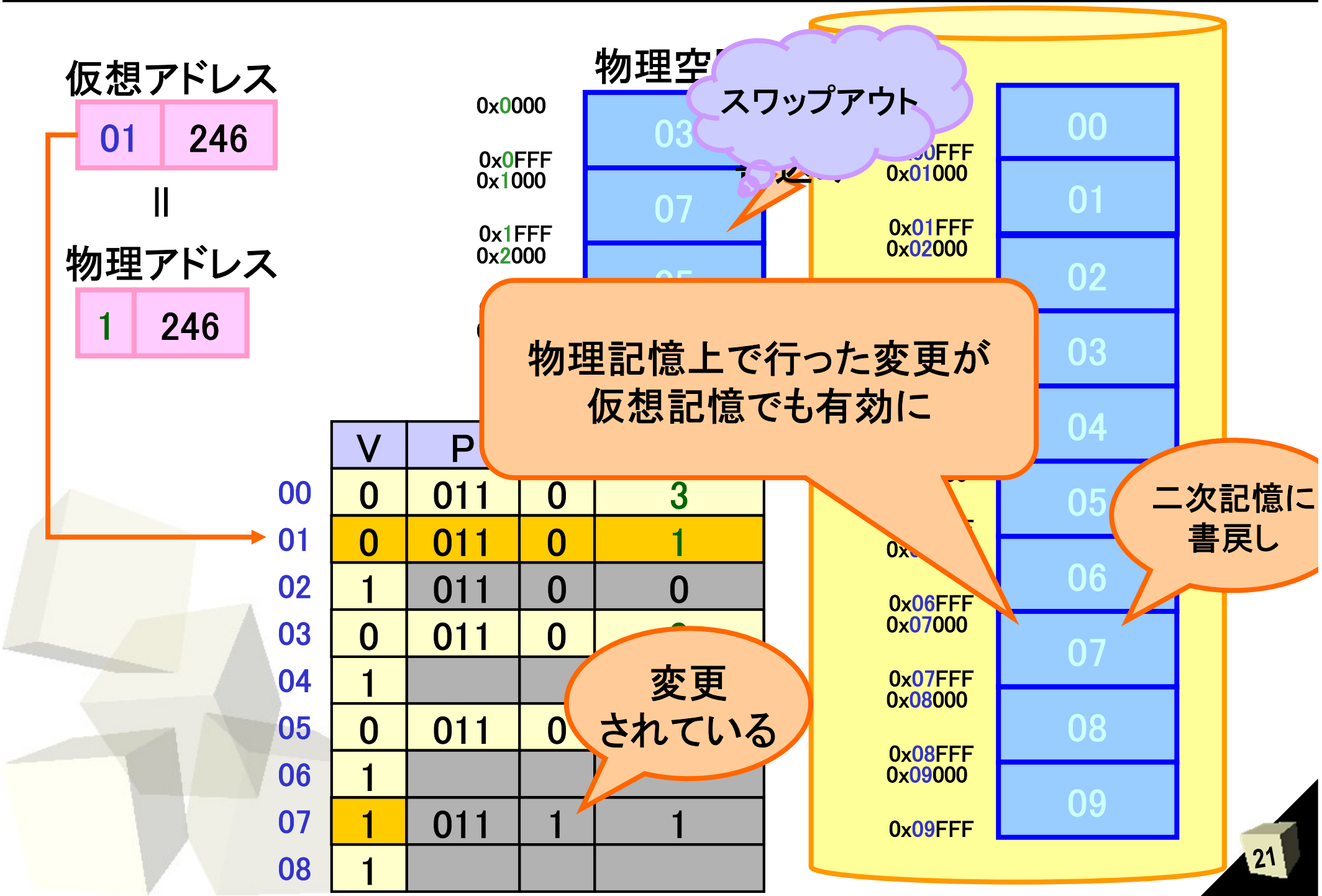
0x0000	02
0x0FFF 0x1000	07
0x1FFF 0x2000	05
0x2FFF 0x3000	00
0x3FFF	

スワップアウト

0x00000	00
0x00FFF 0x01000	01
0x01FFF 0x02000	02
0x02FFF 0x03000	03
0x03FFF 0x04000	04
0x04FFF 0x05000	05
0x05FFF 0x06000	06
0x06FFF 0x07000	07
0x07FFF 0x08000	08
0x08FFF 0x09000	09
0x09FFF	

	V	P	C	ページフレーム
00	0	011	0	3
01	1			
02	1	011	0	0
03	0	011	0	0
04	1			
05	0	011	0	2
06	1			
07	0	011	0	1
08	1			

ページングの動作


仮想アドレス
01
246
||
物理アドレス
1
246
物理空
0x0000
0x0FFF
0x1000
0x1FFF
0x2000
03
07
スワップアウト
0x01000
0x01FFF
0x02000
00
01
02
03
04
05
06
07
08
09
0x06FFF
0x07000
0x07FFF
0x08000
0x08FFF
0x09000
0x09FFF
物理記憶上で行った変更が
仮想記憶でも有効に
二次記憶に
書戻し
変更
されている
V
P
00 0 011 0 3
01 0 011 0 1
02 1 011 0 0
03 0 011 0
04 1
05 0 011 0
06 1
07 1 011 1 1
08 1

ページテーブル

- Vフラグ (Virtual Memory flag)
  - そのページが主記憶上に存在するか否かを示す
  - 「0」の場合は既に主記憶上に存在するので，そのままアクセス可
  - 「1」の場合はスワップインが必要

- Cフラグ (Change flag)
  - スワップイン後，そのフレームに対して書き込みが行われたか（変更されたか）否かを示す
  - 「1」の場合は，スワップアウト時に二次記憶へのフレームの書き戻しが必要

Vフラグ

- V=1であるようなエントリが指定された場合
  - ページフォルト割り込みを発生
    - → プログラムを停止
  - スワップ操作
    - → 必要であればスワップアウト
    - → スワップイン

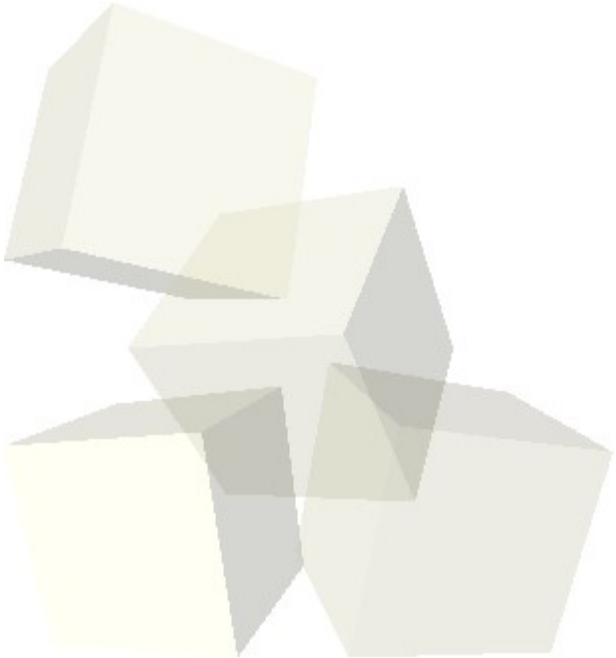
スワップ操作は
必要最低限に
おさえたい

- アクセス速度の違い
  - 主記憶： 10^{-7}秒
  - 二次記憶： 10^{-3}秒

数千倍遅い

Cフラグ

- 当該ページに対して書き込み(変更)が
  あったか否かを示す
  - 変更があった場合はもちろん
    スワップアウト時に二次記憶への書き戻しが必要
  - 変更がなかった場合は,
    スワップアウト時の書き戻しをサボれる



転送コストの
削減

Pフラグ

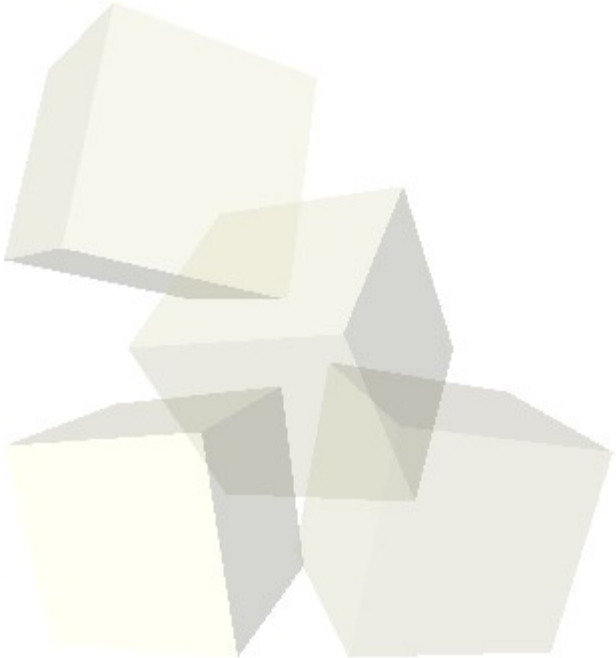
- ページごとにアクセス制御可能
  - 読み出し 可・不可
  - 書き込み 可・不可
  - 実行　　可・不可

- 実行
  - ウィルスなどによる，予期しないアドレスからの実行を防いだりできる
  - 例）スタック領域は実行不可能に
    - → スタックは通常，プログラムを格納しない

復習：フラグメンテーション

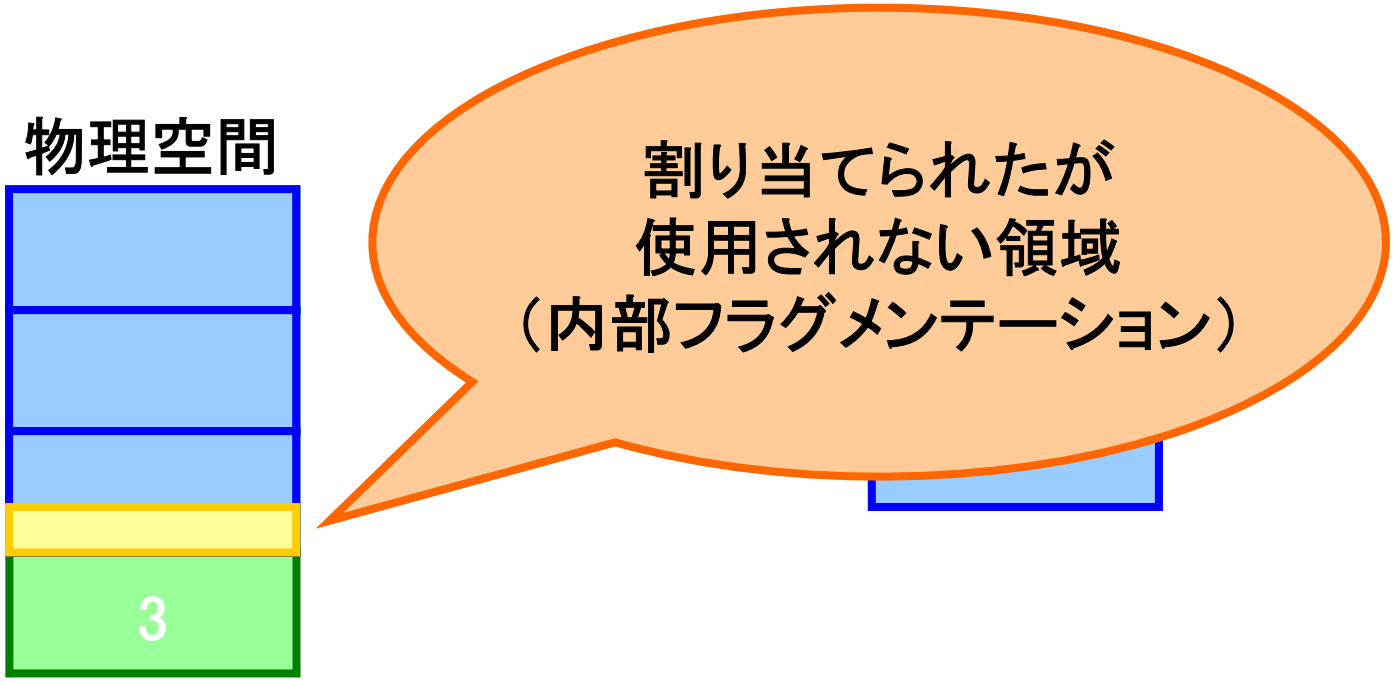
空き領域の断片化

- 処理の進行に伴い，小さな空き領域が増加
  空き領域が断片化（フラグメンテーション）
- 主記憶全体の空き容量が十分でも，新しいプロセスに
  連続した領域を割り当てられない
- 統計的には，全領域数の約 1/3 が無駄に



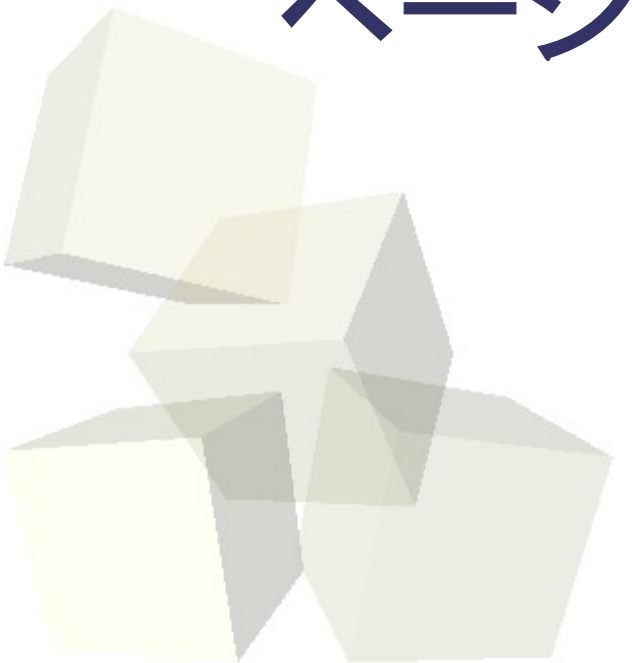
ページングの場合

- 主記憶に割り当てられる領域は，ページ単位
  - 全てのページが何らかのプロセスに割り当てられる



  - 1ページが 4～8kB程度なので，
    主記憶の全容量からすると微々たる大きさ

9.3
ページングの問題点と解決策



ページングの問題点

ページテーブルの巨大さ

- 例）仮想アドレス：32bit，1ページ：8kB の場合
  ページエントリ数：500k個（約50万）
- ページテーブルはプロセス毎に独立
  例）100プロセスの場合エントリ数：約5000万
  1エントリ10Bで構成すると500MB

メモリアクセスの増大

- ページテーブルは主記憶内に存在
- 1回で2度の主記憶アクセスが必要
  → ページテーブルへのアクセス
  → 物理アドレスへのアクセス

ページングの問題点と解決法

- ページテーブルの巨大さ

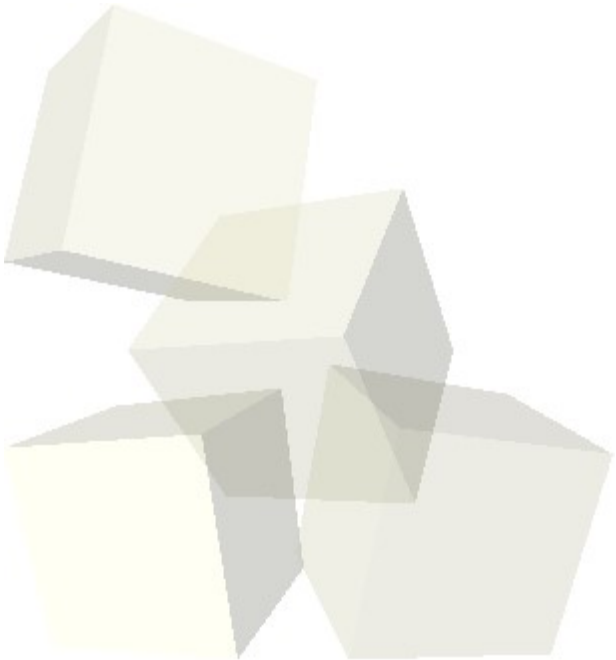
ハッシュ関数によるページテーブル

- メモリアクセスの増大

連想レジスタ方式

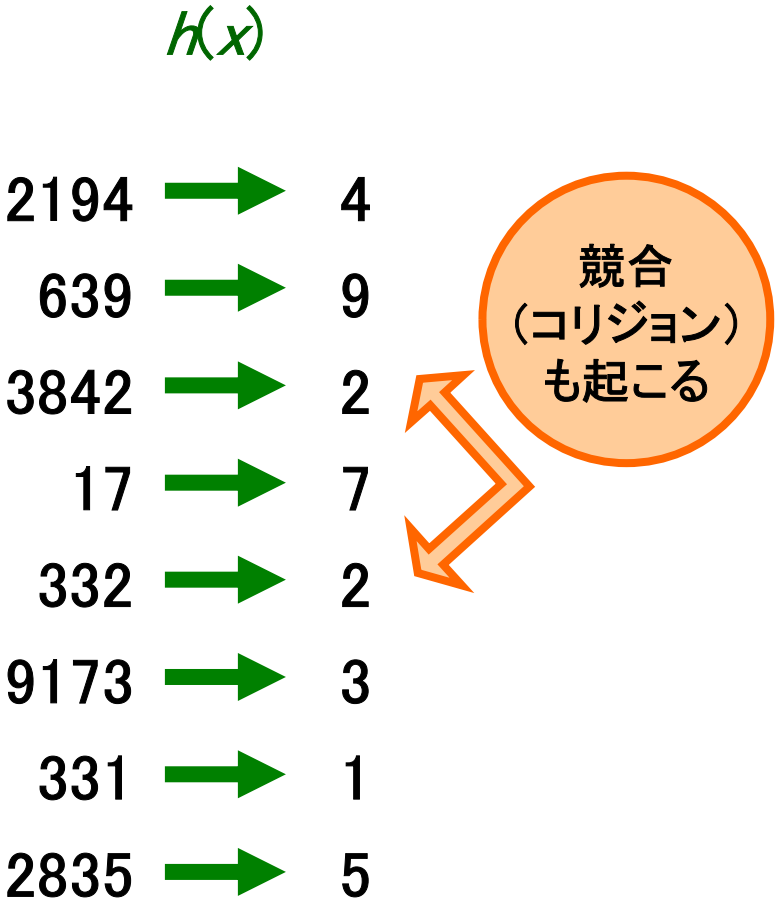
ハッシュ関数によるページテーブル

…のまえに



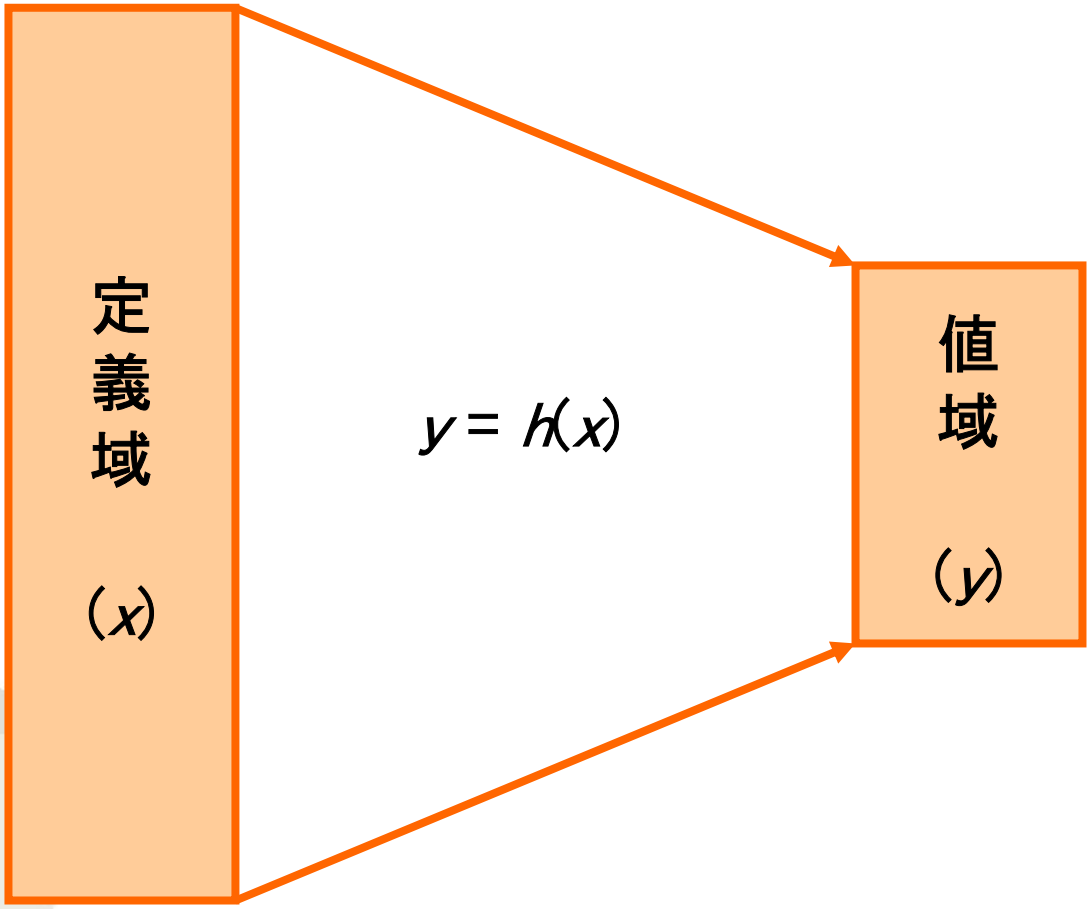
ハッシュ関数

- データを一定長のデータに要約するための関数
  - 一種の圧縮
  - 「要約」するので，
    データの損失が起こる
    （不可逆変換）
  - 例）数値 x を1桁に要約
    するハッシュ関数
    $h(x) = x \% 10;$



ハッシュ関数

- ある範囲(定義域)のデータを,
  ある範囲(値域)の値に要約する写像
  - 定義域 > 値域



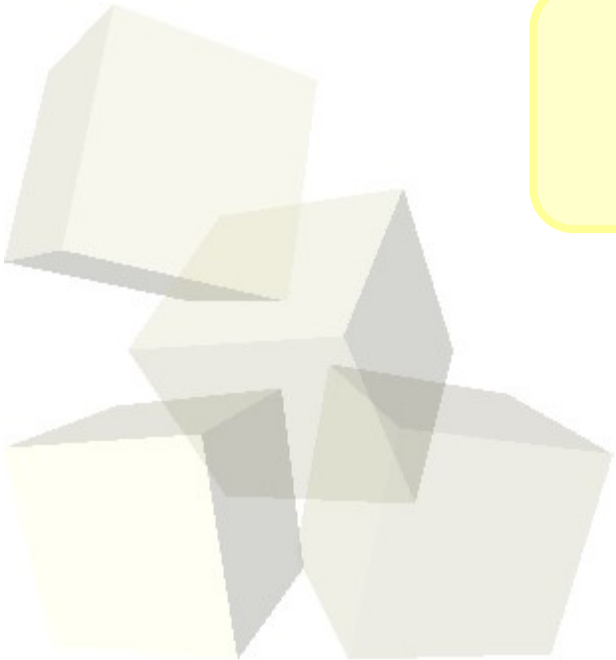
ページングの問題点と解決法

- ページテーブルの巨大さ

ハッシュ関数によるページテーブル

- メモリアクセスの増大

連想レジスタ方式



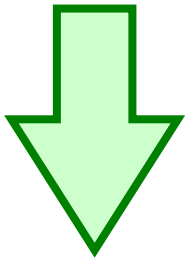
ハッシュ関数によるテーブル

- ページテーブルが大きくなる原因
  - （仮想記憶の大きさと同じエントリ数）
    ×（同時実行プロセス数）
    だけのエントリ数が必要

- ページテーブル内に格納されるフレームは
  - Vフラグが0なら主記憶上のアドレスを指す
  - Vフラグが1なら仮想記憶上（二次記憶）のアドレスを指す

ハッシュ関数によるテーブル

- V=1 なエントリの場合
  - ページフォルト
  - スワップイン（10^{-3}秒）
    - → ページテーブルを主記憶上に置いて高速アクセス（10^{-7}秒）可能にしても，結局は大きい処理時間がかかってしまう



  - V=1なエントリを主記憶に置く必要はない
  - 二次記憶上に置いても全体性能に大きな影響はない

ハッシュ関数によるテーブル

仮想アドレス

06	246

||

物理アドレス

2	246

ハッシュ関数
h(x) = x % 4;

物理空間

0x0000
0x0FFF
0x1000
0x1FFF
0x2000

03
01
06

0x00000
0x00FFF
0x01000
0x01FFF
0x02000
0x02FFF
0x03000
0x03FFF
0x04000
0x04FFF

00
01
02
03
04
05
06
10

共有されるから，このエントリが
どのページ番号に関するデータを
保持しているか
記憶しておく必要がある

物理記憶と
同じサイズ
（小容量）

ントリは，
ハッシュ結果が02となる
「02」「06」「10」...
などで共有される

	flag	ページ番号	ページフレーム	ポインタ
00		00	3	
01		01	1	
02		06	2	
03		03	0	

ハッシュ関数によるテーブル

仮想アドレス

07	357

||

物理アドレス

3	357

ハッシュ関数
h(x) = x % 4;

物理空間

0x0000
0x0FFF
0x1000
0x1FFF
0x2000
0x2FFF
0x3000
0x3FFF

03
01
06
00

スワップアウト

0x00000
0x00FFF
0x01000
0x02FFF
0x03000
0x03FFF
0x04000
0x04FFF
0x05000
0x08FFF
0x09000
0x09FFF

00
01
02
03
04
05
08
09
10

	flag			ポインタ
00				
01		01		
02		06	2	
03		03	0	00

次エントリへのポインタ

このエントリを使いたいが，03の情報を潰してしまうのはマズい

ハッシュ関数によるテーブル

仮想アドレス

11	357

||

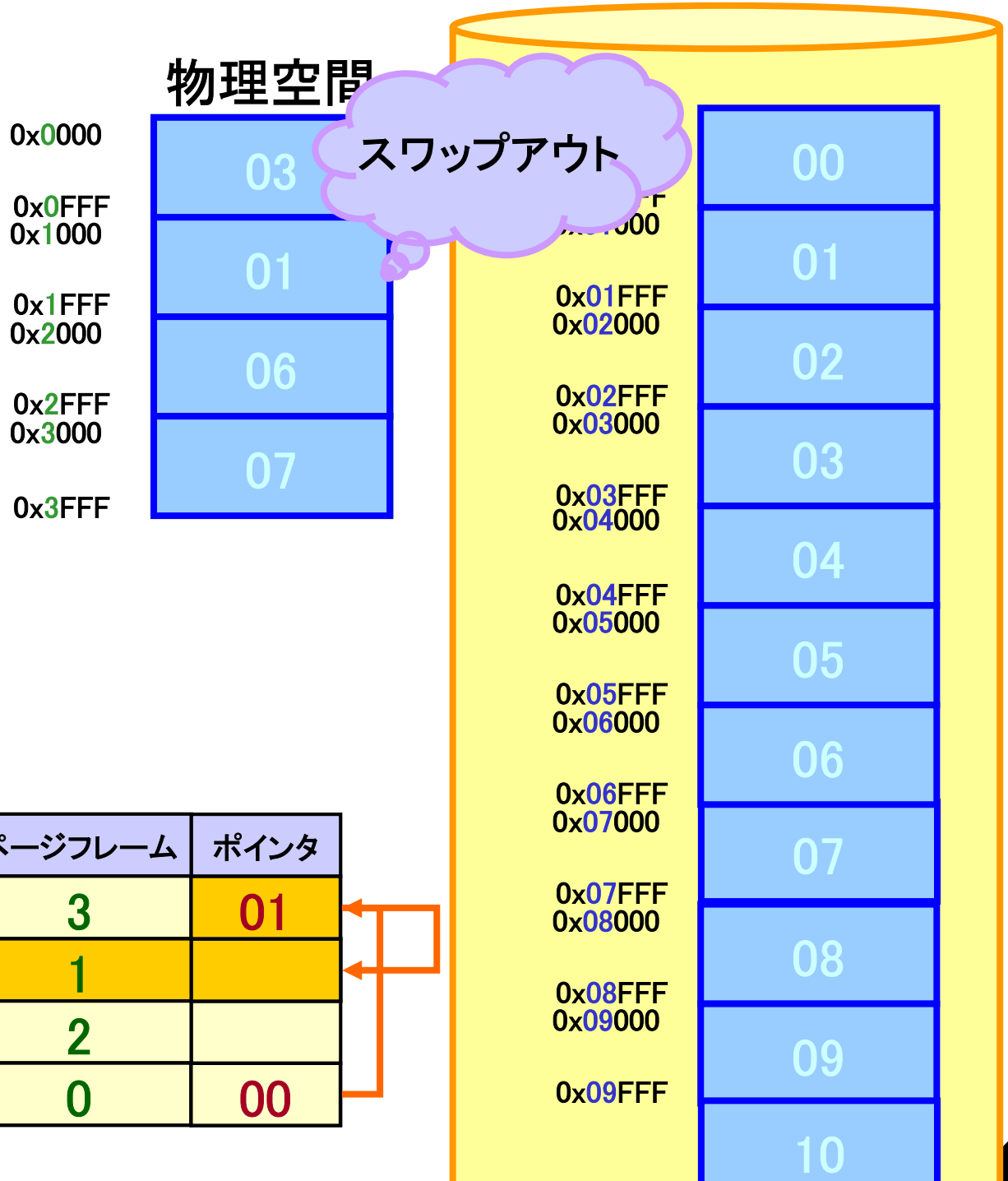
物理アドレス

1	357

ハッシュ関数
h(x) = x % 4;

	flag	ページ番号	ページフレーム	ポインタ
00		07	3	01
01		11	1	
02		06	2	
03		03	0	00

物理空間



ページテーブルの大きさ

- 改良前
  - 仮想アドレス空間のフレーム数と
    同じエントリ数
- ハッシュ関数によるページテーブル
  - 物理アドレス空間のページフレーム数と
    同じエントリ数

ハッシュ関数が持つべき条件

変換結果の値域への均一分散

- ハッシュ関数による変換結果の値が偏らない必要
  - → 変換結果が衝突することをコリジョンと呼ぶ
- 偏ると，検索時にポインタをたどる確率が高くなり，オーバヘッド増大

変換の高速性

- ハッシュ関数による変換は主記憶アクセスの度に発生
- 変換が低速であると，そもそも意味がない

ページングの問題点と解決法

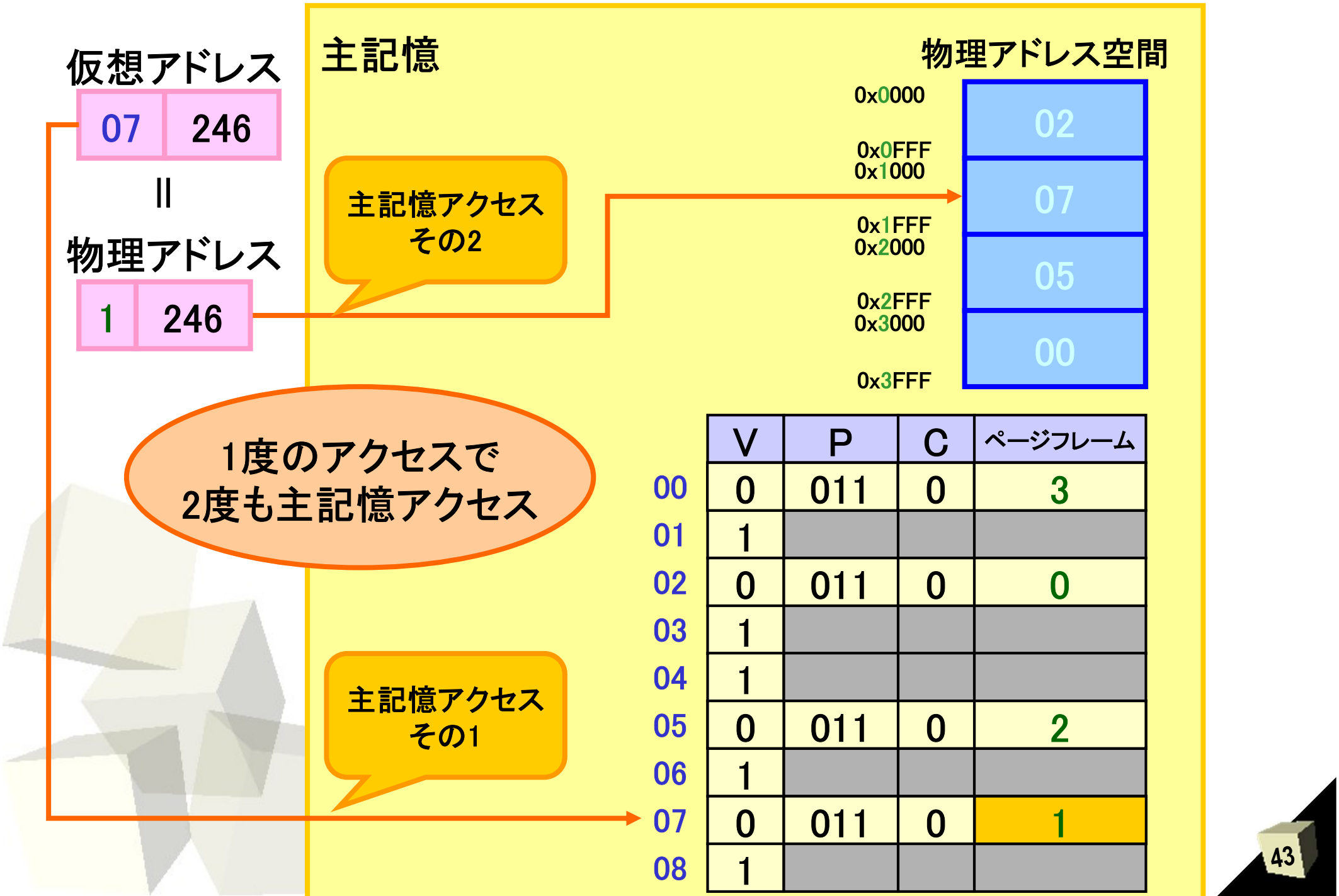
- ページテーブルの巨大さ

ハッシュ関数によるページテーブル

- メモリアクセスの増大

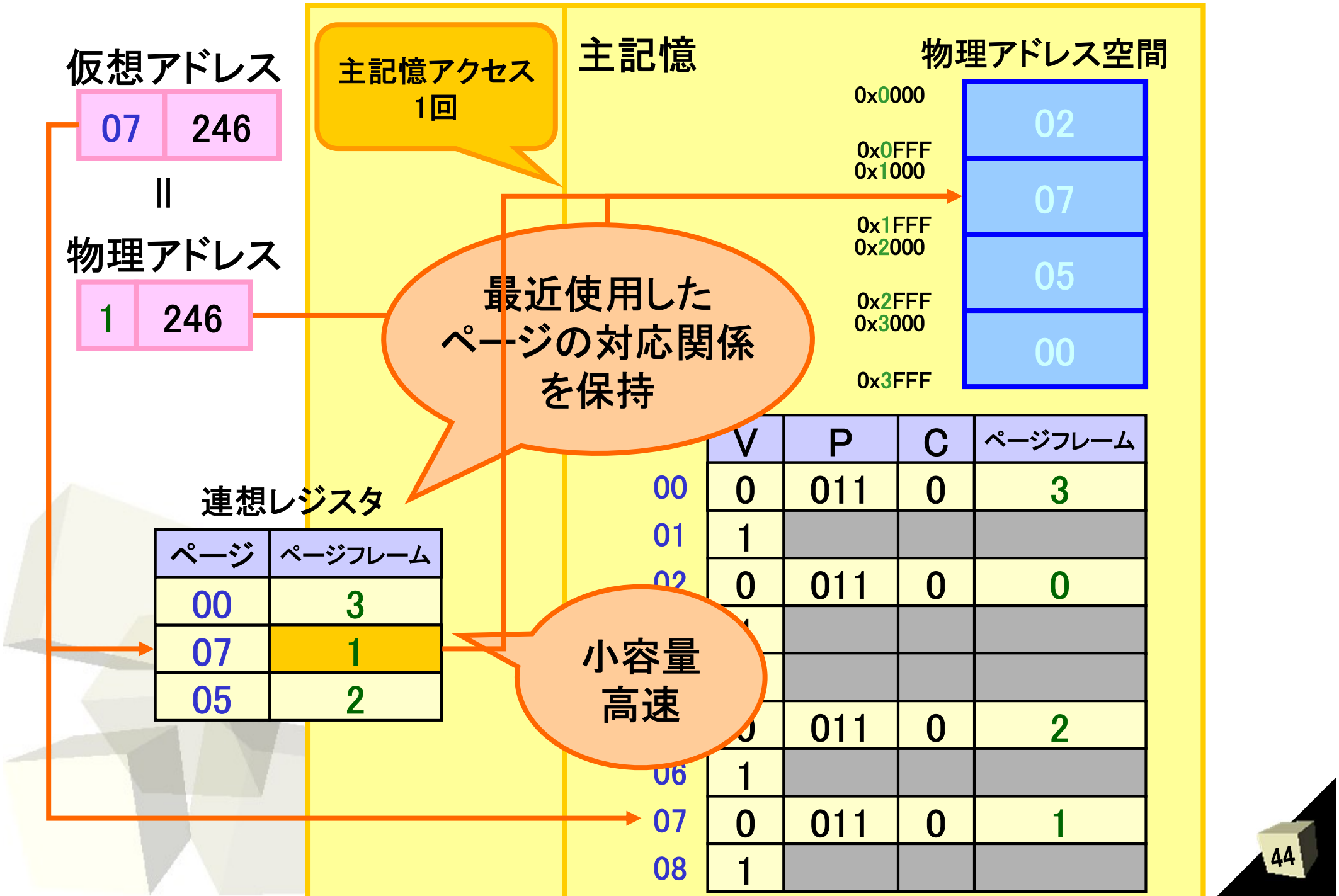
連想レジスタ方式

ページテーブルの参照



	V	P	C	ページフレーム
00	0	011	0	3
01	1			
02	0	011	0	0
03	1			
04	1			
05	0	011	0	2
06	1			
07	0	011	0	1
08	1			

連想レジスタ方式


仮想アドレス
07
246
=
物理アドレス
1
246
主記憶アクセス
1回
主記憶
物理アドレス空間
0x0000
0x0FFF
0x1000
0x1FFF
0x2000
0x2FFF
0x3000
0x3FFF
02
07
05
00
最近使用した
ページの対応関係
を保持
連想レジスタ
ページ
ページフレーム
00
3
07
1
05
2
小容量
高速
V
P
C
ページフレーム
00
0
011
0
3
01
1
02
0
011
0
0
06
1
07
0
011
0
1
08
1

連想レジスタ方式

- 連想レジスタ
  (TLB: Translation Lookaside Buffer)
  - 最近行われた変換結果をCPU内で保持
  - 小容量で高速
  - 一般的にプログラムは,
    「一度アクセスしたアドレスを
    近いうちに再度アクセスする可能性が高い」
    ことを利用

今日のまとめ

ページング

- 主記憶の再配置システム
- 仮想アドレスの上位(ページ番号)を
  物理アドレスの上位(ページフレーム番号)に
  変換することでアドレス変換
- フラグメンテーションも改善

ページテーブル

- 上記変換を行うためのテーブル
- ページに対応するエントリごとにフラグなどを配置
  - → ページへのアクセス権などを設定可能

今日のまとめ

ページングの改善法

- ハッシュ関数
  - → ページテーブルの巨大さを緩和
  - → 仮想アドレス空間に比例する大きさが必要だったのが，物理アドレス空間に比例する大きさまで縮小
- 連想レジスタ
  - → 主記憶へのアクセス回数の緩和
  - → 最近使用した「ページ番号→ページフレーム番号」の変換結果をCPU内で記憶
  - → ページテーブル検索のための主記憶アクセスを削減