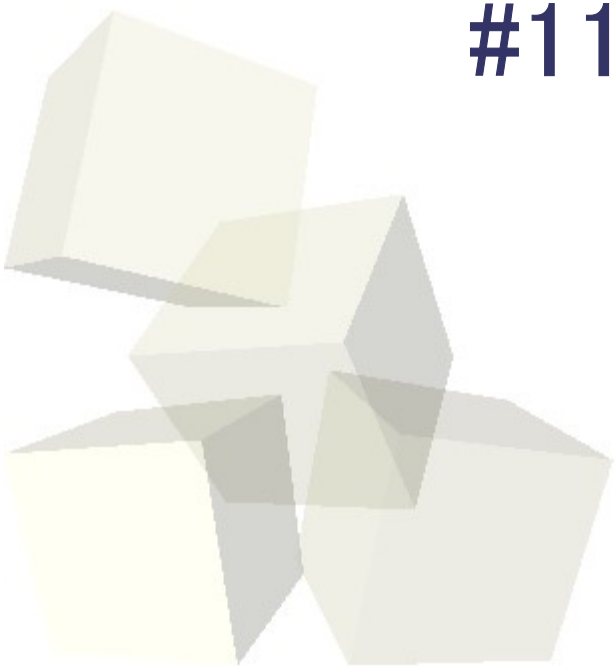
オペレーティングシステム

#11 主記憶管理：仮想記憶



復習：主記憶管理

- OS領域とプログラム領域の分離
  - 下限レジスタ機構
  - キー／ロック機構
- プロセスのための領域確保
  - ベストフィット
  - ファーストフィット
  - ワーストフィット
- プロセスに割り当てた領域の管理
  - リスト方式
  - ビットマップ方式

復習

主記憶の動的再配置

- ページング
  - → ○ 固定長ページによるフラグメンテーションの解決
  - → × ページテーブル自体が大きな主記憶領域を消費
- セグメンテーション
  - → ○ 割り当て領域の大きさが可変
  - → ○ 複数の領域を1プロセスに割り当て可
  - → × フラグメンテーション
- ページ化セグメンテーション
  - → セグメントをページ単位で扱う
  - → ページングとセグメンテーションの利点の融合

復習

ページ化セグメンテーション

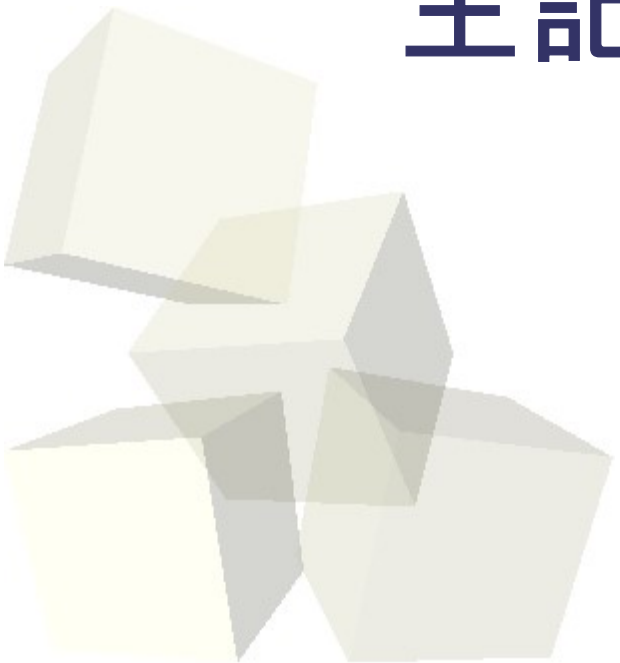
- セグメンテーションの利点を継承
  - → 割り当て領域の大きさが可変
  - → 複数の領域を1プロセスに割り当て可
- フラグメンテーションの回避
  - → 主記憶割り当ては基本的にページ単位
- ページテーブルの分散
  - → ページテーブルが複数に分割されるので，
    多重レベルページング同様，その一部を
    仮想記憶に追い出すことで主記憶使用量削減

今日の内容

- 仮想記憶の具体的な実現
  - どのタイミングでスワップ処理をするか
  - どのページをスワップアウトさせるか
  - etc...

- の前に...
  - そもそも主記憶へのアクセスはどのような特徴を持つかをまず説明(11.3)
  - その特徴をふまえて，具体的な実現方法を概説

11.3
主記憶アクセスの局所性



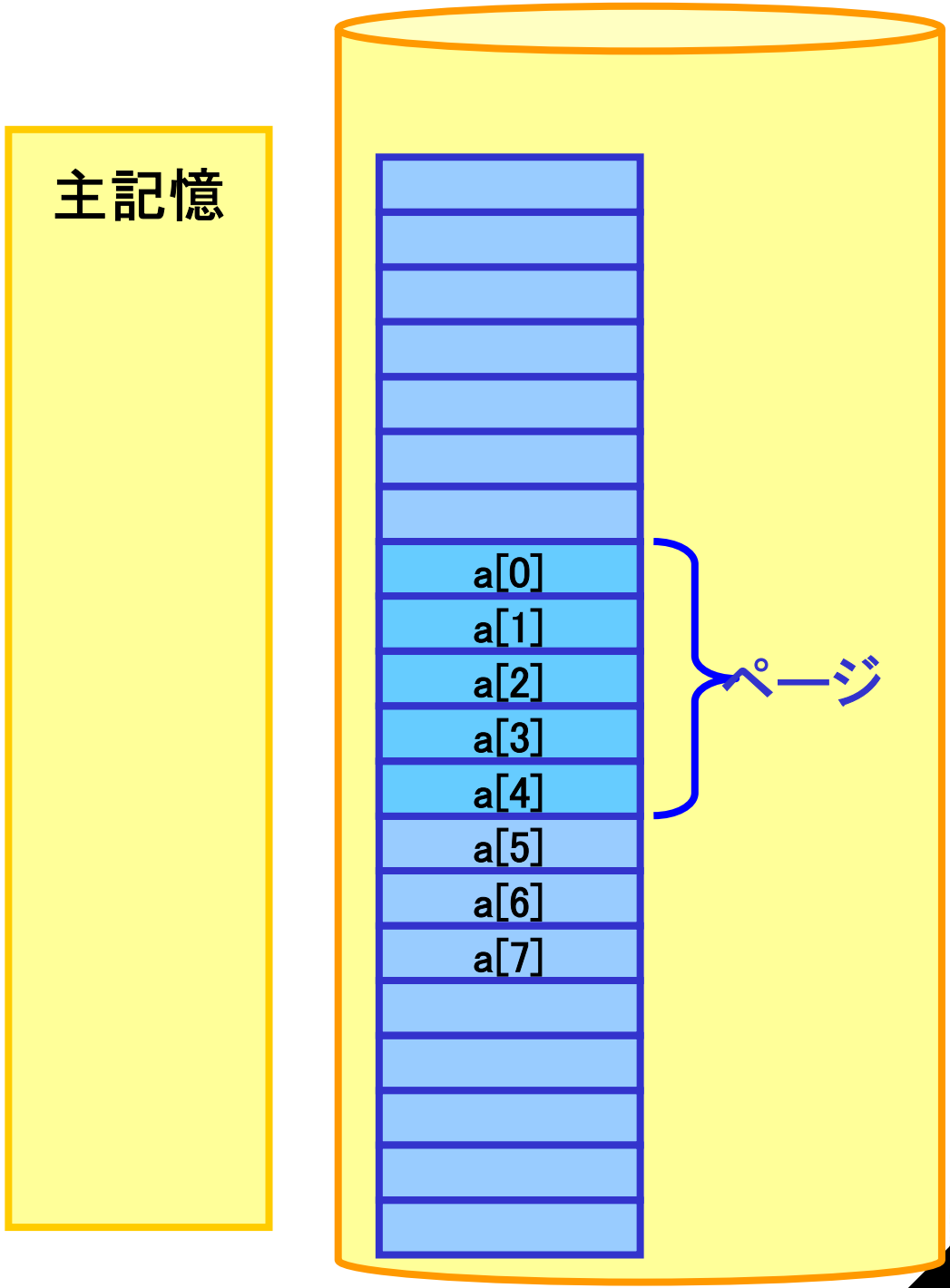
主記憶アクセスの局所性

- 主記憶アクセスには，ある特徴がある
  - ノイマン型コンピュータ
- 空間的局所性
  - あるアドレスにアクセスが発生した場合，
    次はその近くのアドレスにアクセスされる可能性が
    高い
- 時間的局所性
  - あるアドレスにアクセスが発生した場合，
    近いうちには同じアドレスにアクセスされる
    可能性が高い

空間的局所性

- 空間的局所性
  - 主記憶上のある場所が参照されると，(近いうちに)その近傍も参照される可能性が高い

```
for(i=0; i<n; i++){
        sum += a[i];
}
```



時間的局所性

- 時間的局所性
  - ある短い時間だけ見た場合，プログラムがアクセスするページは限られている
  - 最近アクセスされたページは，近いうちに再びアクセスされる可能性が高い

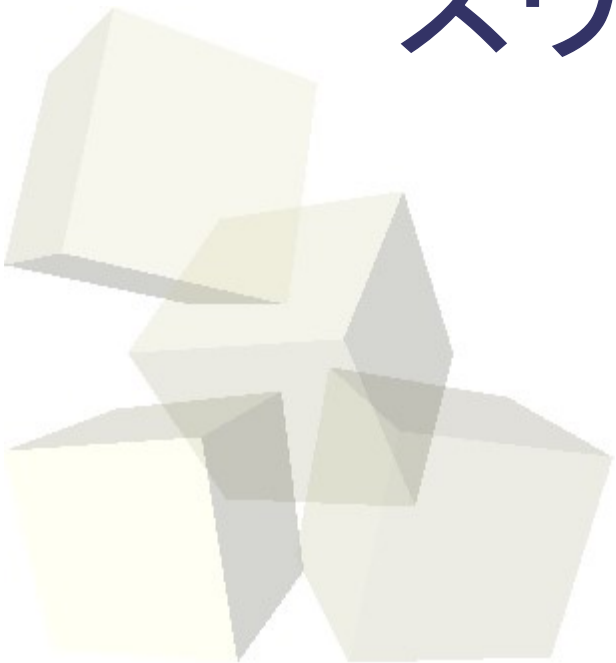
```
for(i=0; i<n; i++){
        sum += a[i];
}
```

逆に言うと，
この for文 を抜けた瞬間，
アクセスされるページは
大幅に変わる可能性がある
（フェーズ化）

- 局所性の原因となるプログラムの特徴
  - 関数を用いてプログラムを構造化
    - → 関数内ではアクセスする記憶領域がほぼ同じ（時間的＆空間的）
  - プログラムの大部分はループから成る
    - → ループイタレータなど，同一変数へのアクセス（時間的）
    - → 配列処理など，連続領域へのアクセス（空間的）

- これらの特徴をふまえて
  仮想記憶の効率的な実装方法について考える

11.1
スワップスケジューリング



スワップスケジューリング

- 仮想記憶
  - ○ 大きさが無限の仮想アドレスを提供
  - × スワップ操作に膨大な時間を要する
    - → 2次記憶（ディスク）へのアクセスは，
      主記憶アクセスに比べて非常に遅い

- スワップ操作
  - 可能な限り低頻度で
  - 可能な限り低コストで

以下，
低頻度かつ低コスト
になる方法を探る

スワップスケジューリング

- 以後，スワップ操作を3要素に分けて考える

スワップを行うタイミング

スワップインすべき場所の選択

スワップアウトすべきページの選択

スワップスケジューリング

スワップを行うタイミング

スワップインすべき場所の選択

スワップアウトすべきページの選択

スワップを行うタイミング

- デマンドページング
  - ページフォルトが発生した時点
  - 必要になった際に必要なページをスワップイン
  - スワップインの前にページフォルトが必ず発生
    - → あたりまえ
  - ページフォルト処理にかかるコストを削減したい
- プリページング（予測ページング）
  - 必要になりそうなページを前もってスワップイン
  - 予測があたればページフォルトは発生せずコスト削減

プリページング手法

- デマンドプリフェッチ
  - ページフォルト割込が発生したタイミングで
    - → ページフォルトを起こした対象ページは勿論スワップイン
    - → 将来必要と予想される数ページを同時にスワップイン
      - 例えば対象ページの近傍ページ

空間的局所性
に基づく考え方

- 初期ロードプリフェッチ
  - プログラムが最初にロードされるタイミングで
    - → プログラムの実行開始直後は，ページフォルトが多発
    - → プログラム開始直後は多くの近傍ページを同時にスワップインしておいたほうがよい
      - どうせプログラム全体が読み込まれるはずなので

スワップスケジューリング

スワップを行うタイミング

スワップインすべき場所の選択

スワップアウトすべきページの選択

スワップインすべき場所

- 結論

どこにスワップインしても
性能に影響は出ない

スワップスケジューリング

スワップを行うタイミング

スワップインすべき場所の選択

スワップアウトすべきページの選択

復習：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

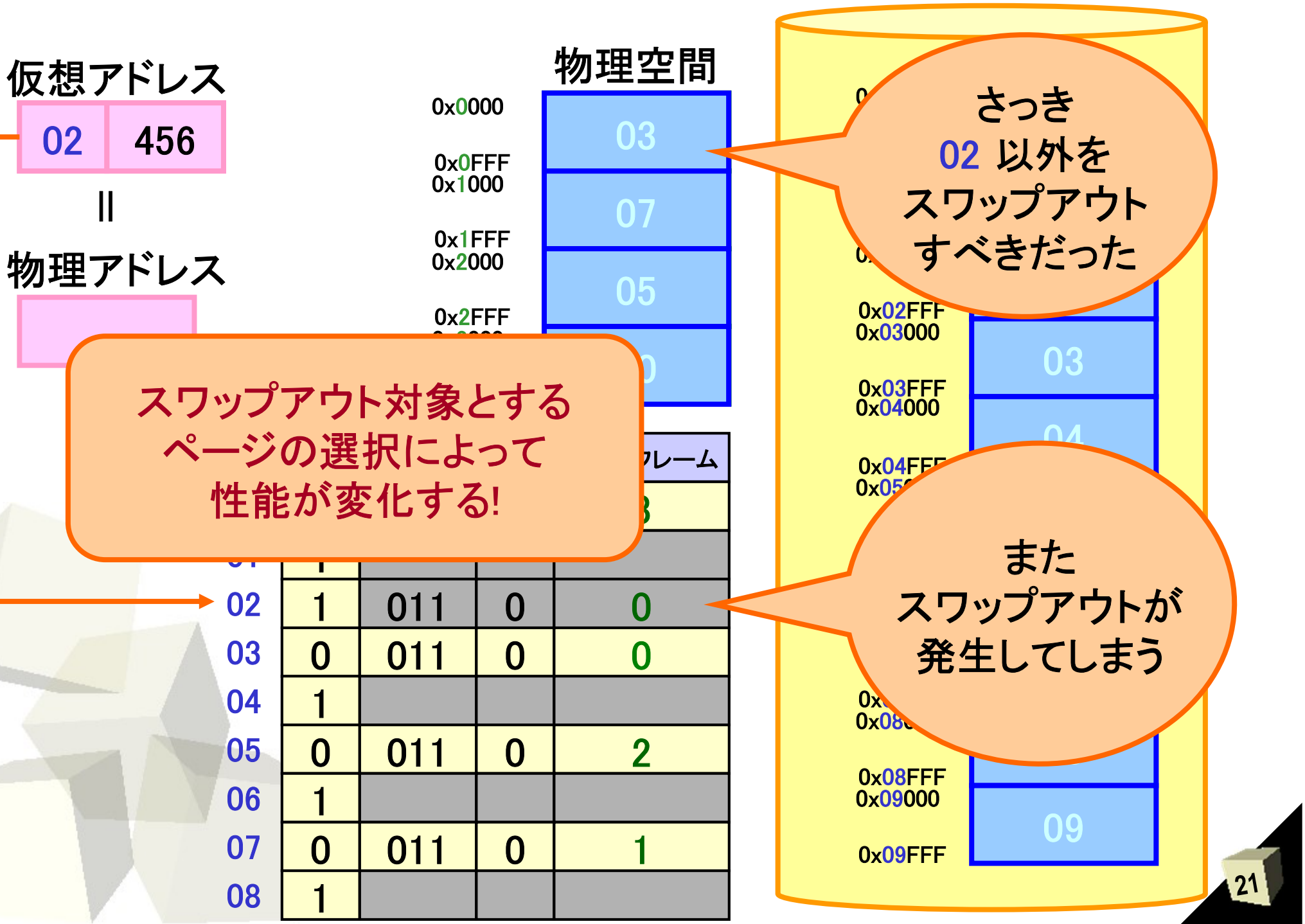
0x0000	02
0x0FFF 0x1000	07
0x1FFF 0x2000	05
0x2FFF 0x3000	00
0x3FFF	

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			

スワップアウト


仮想アドレス
02
456
=
物理アドレス
物理空間
0x0000
0x0FFF
0x1000
0x1FFF
0x2000
0x2FFF
03
07
05
0
スワップアウト対象とする
ページの選択によって
性能が変化する!
フレーム
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
さっき
02 以外を
スワップアウト
すべきだった
0x02FFF
0x03000
0x03FFF
0x04000
0x04FFF
0x08FFF
0x09000
0x09FFF
03
09
また
スワップアウトが
発生してしまう

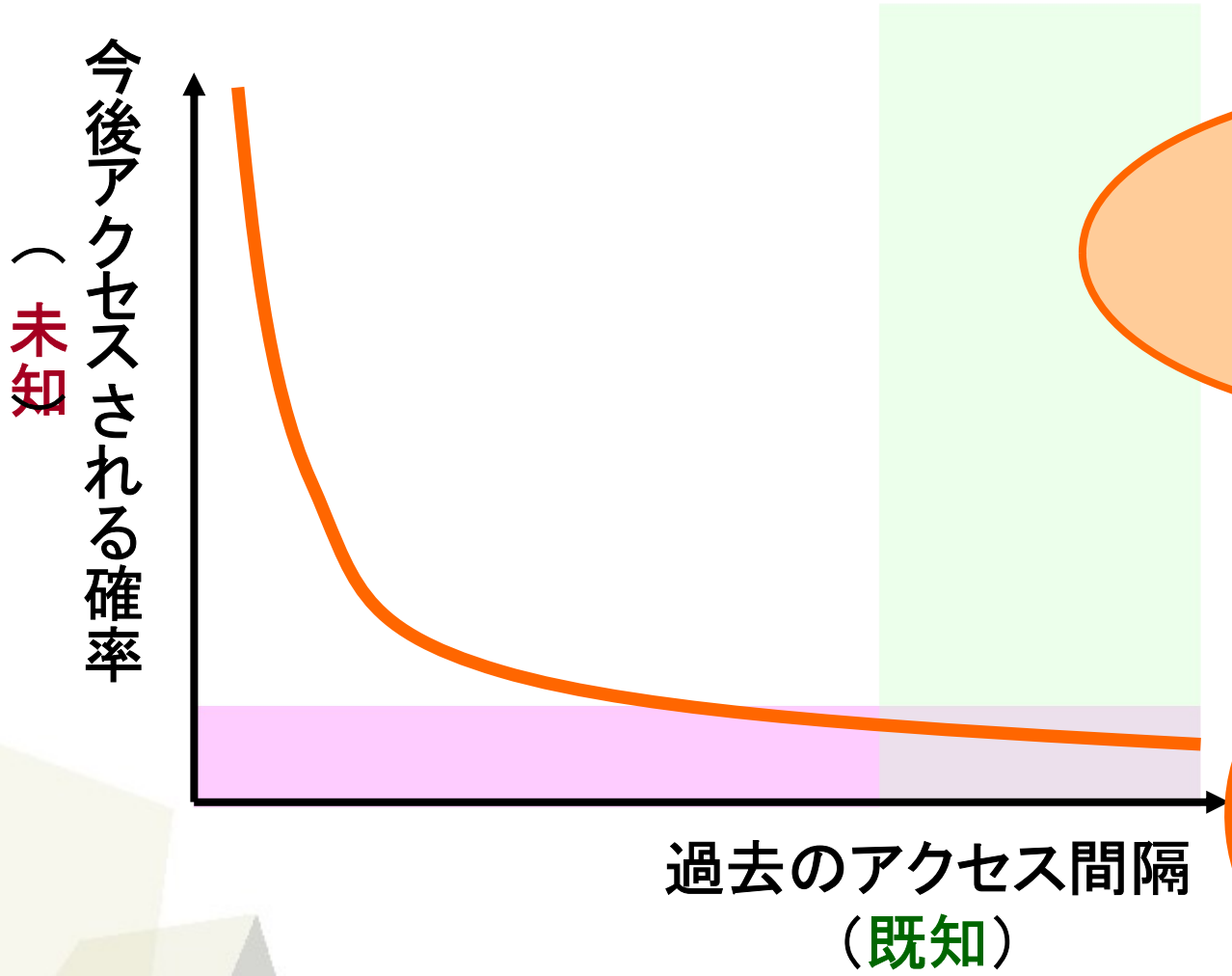
スワップアウト対象の選択

- スワップアウトすべきページ
  - 次にアクセスされる可能性が最も低いページ
  - 次回アクセスされるまでの時間が最も長いページ

これらは
前もって正確に知ることはできない

- 近似的に知る方法を考える

LRU

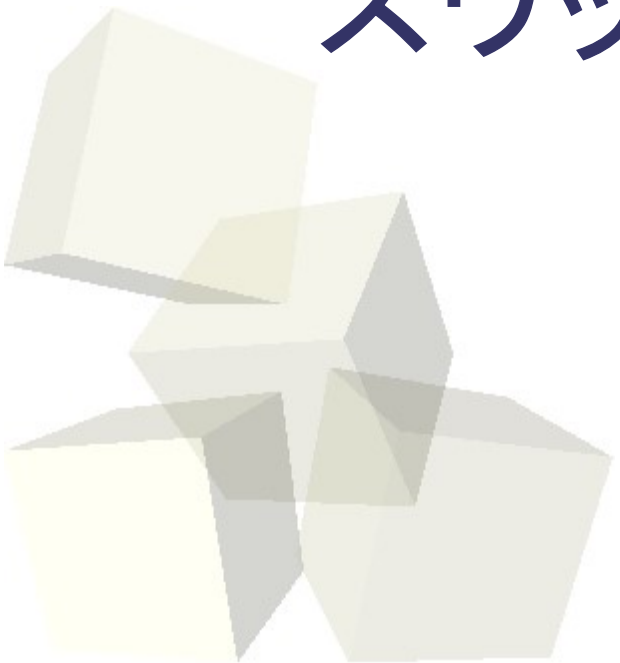


時間的局所性
に基づく考え方

LRU
Least Recently Used

- アクセス確率の低いページを選びたい
- アクセス間隔の長いページを選べばよい

11.2
参照ビットを用いた
スワップアウト対象ページの
決定



LRUの（不可能な）実現方法

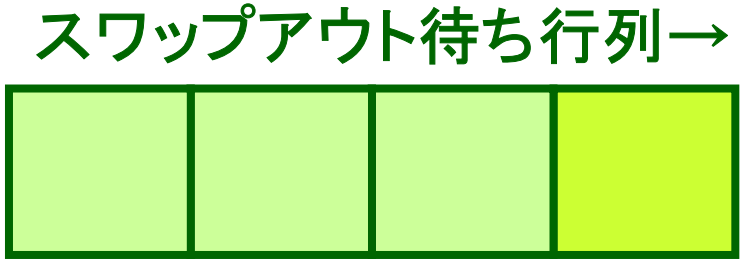
- LRUを実現するには...
  - ページテーブルに，各ページのアクセス時刻を記録する項目を追加
  - 主記憶アクセスごとに
    - → 現在時刻を調べ
    - → アクセス時刻を更新
  - ページフォルト時に
    - → ページテーブルを走査して
    - → 最もアクセス時刻の古い項目を探す

コストが大きすぎて
本末転倒

参照ビットを用いた近似LRU

- 参照ビット
  - 当該ページがアクセスされたか否かを表すフラグ
- スワップアウトの待ち行列

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



参照ビットを用いた近似LRU

仮想アドレス

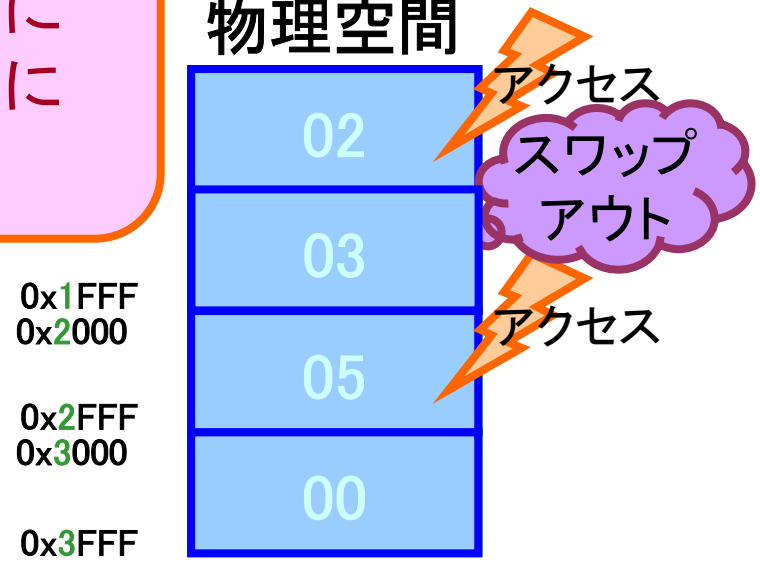
03	101

ページフォルト

待ち行列内で
1のものを後方に
0のものを前方に
シフト

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

物理空間



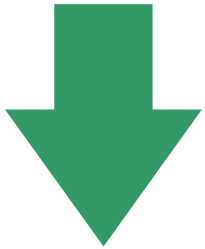
スワップアウト待ち行列→

00	05	07	02

つぎスワップアウトされる

参照ビットを用いた近似LRU

- Rビットが0のものを前方シフト
  - アクセスのなかったページを優先的にスワップアウト
- Rビットが1のものを後方シフト
  - アクセスのあったページをスワップアウトしにくく



- 待ち行列
  - 近似的に，アクセス頻度が低い順に並ぶ

近似LRUの改良案１

- 待ち行列を参照カウンタに
  - 参照があったか否かだけでなく，
    過去に参照があった回数を記憶
  - 最も参照回数の少なかったページをスワップアウト

  - ○ より精密な近似ができる
  - × カウンタ更新のコストが追加
  - × スワップアウトページの検索コスト大
    （全てのカウンタを比較しないといけない）

近似LRUの改良案２

- 更新タイミングの削減
  - 待ち行列内のシフトと参照ビットのクリアの頻度を変える
  - ページフォルト時よりも高頻度に
  - ○ 近似がより精密に
  - × 更新コストの増大

今日のまとめ

- 仮想記憶処理のオーバヘッド削減
  - ページフォルトの発生回数を減らしたい
    - → プリページング
      - 必要となりそうなページを前もってスワップイン
  - スワップアウトの発生回数を減らしたい
    - → スワップアウト対象ページの効率的選択
      - この先，最も参照されなさそうなページをスワップアウト

これらを予測するためには
主記憶アクセスの特徴を把握する必要あり

今日のまとめ

- 主記憶アクセスの特徴 ＝ 局所性

```
for(i=0; i<n; i++){
        sum += a[i];
}
```

- 空間的局所性
  - あるアドレス・ページにアクセスが発生した場合，
    次はその近くのアドレス・ページにアクセスされる
    可能性が高い
- 時間的局所性
  - あるアドレス・ページにアクセスが発生した場合，
    近いうちには同じアドレス・ページにアクセスされる
    可能性が高い

今日のまとめ

仮想記憶処理のオーバヘッド削減

- ページフォルトの発生回数を減らしたい
  - → プリページング
    - 必要となりそうなページを前もってスワップイン

最近アクセスされたページの
近くにあるページ

空間的局所性
から

- スワップアウトの発生回数を減らしたい
  - → スワップアウト対象ページの効率的選択
    - この先，最も参照されなさそうなページをスワップアウト

最近アクセスされていない
(Least Recently Usedな)ページ

時間的局所性
から

今日のまとめ

- LRUの（近似的）実装方法
  - ページテーブルに，参照ビットを付加
  - スワップアウト待ち行列を追加
  - 参照ビットがオン（＝参照された）のページを待ち行列の後方へ
    - → スワップアウトされにくくなる