オペレーティングシステム

第10回

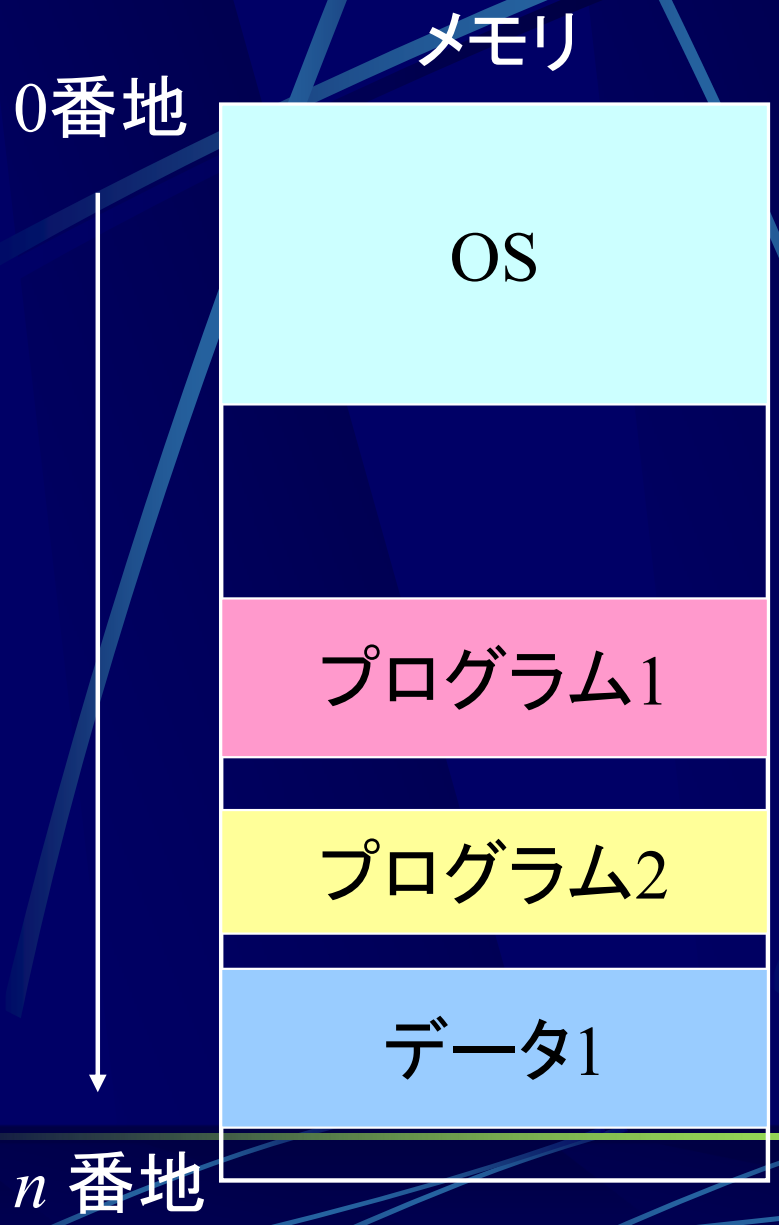
仮想記憶管理(2)

http://www.info.kindai.ac.jp/OS

E号館3階E-331 内線5459

takasi-i@info.kindai.ac.jp

メモリ

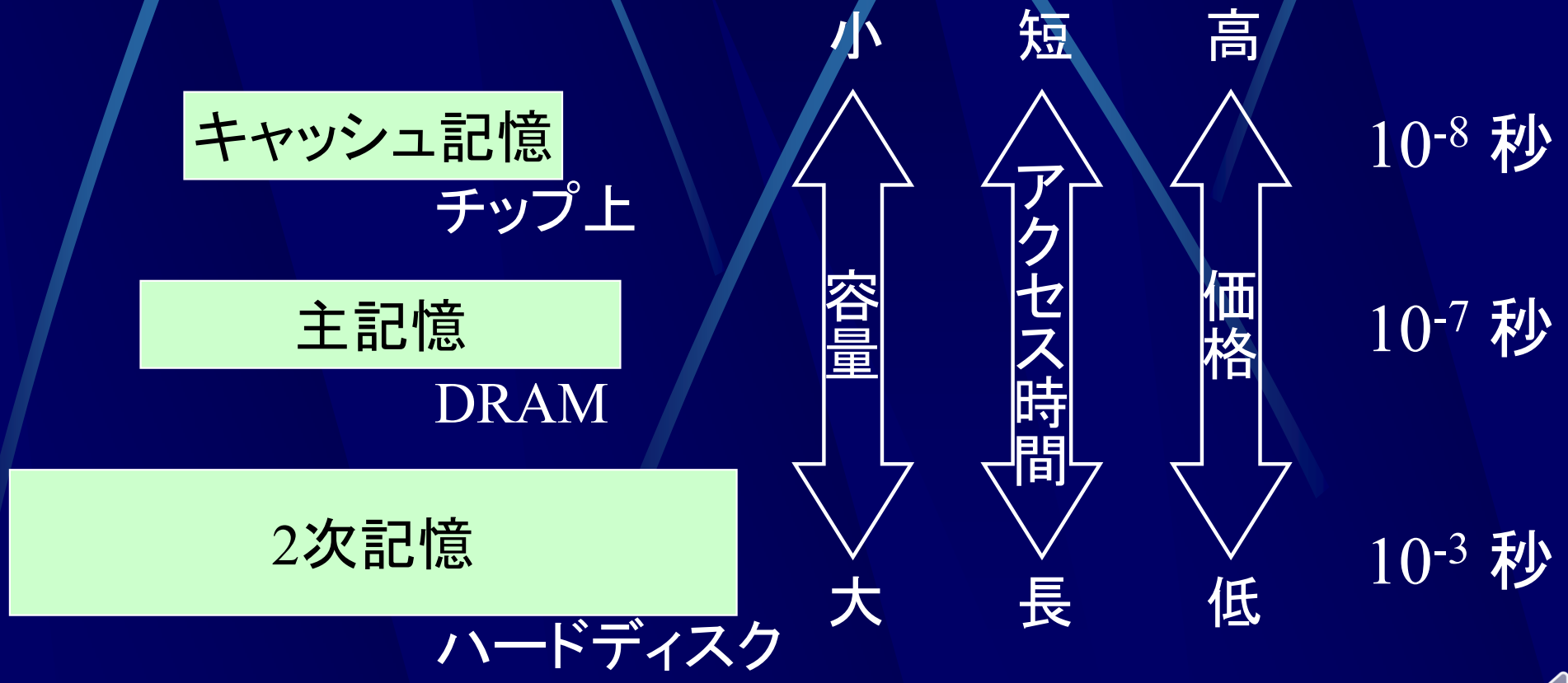


OS, ユーザプログラム, データ
メモリ上に置かれる

メモリ上の位置は
1 次元のアドレスで管理

メモリ

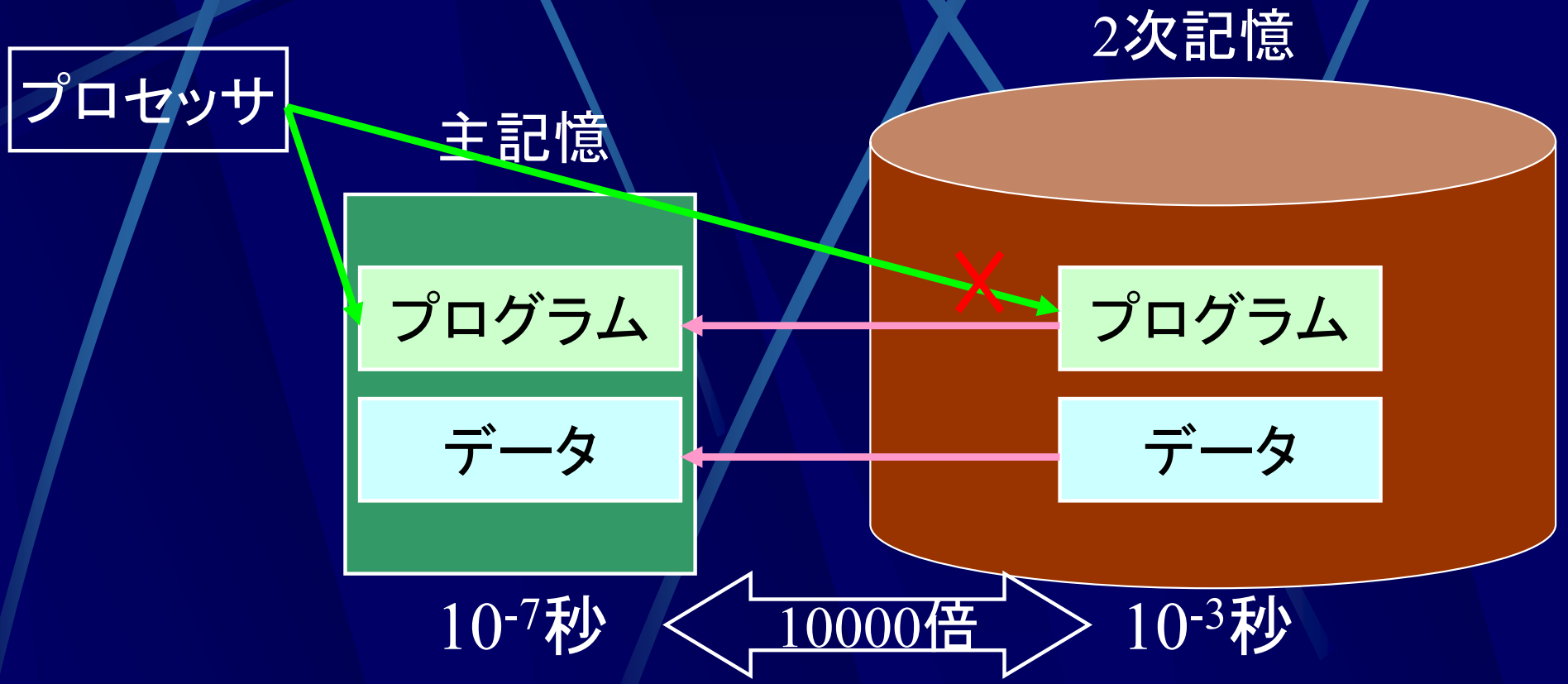
- メモリの記憶階層



メモリ

記憶装置	本, 資料	特徴
キャッシュ (cache memory)	手で保持 	すぐ読める ごくわずかな量しか持てない
主記憶 (main memory)	作業机 	座ったまますぐ手に取れる 置ける量は限られる
2次記憶 (secondary memory)	倉庫 	部屋を出て取りにいく必要あり 大量に置ける

主記憶と2次記憶

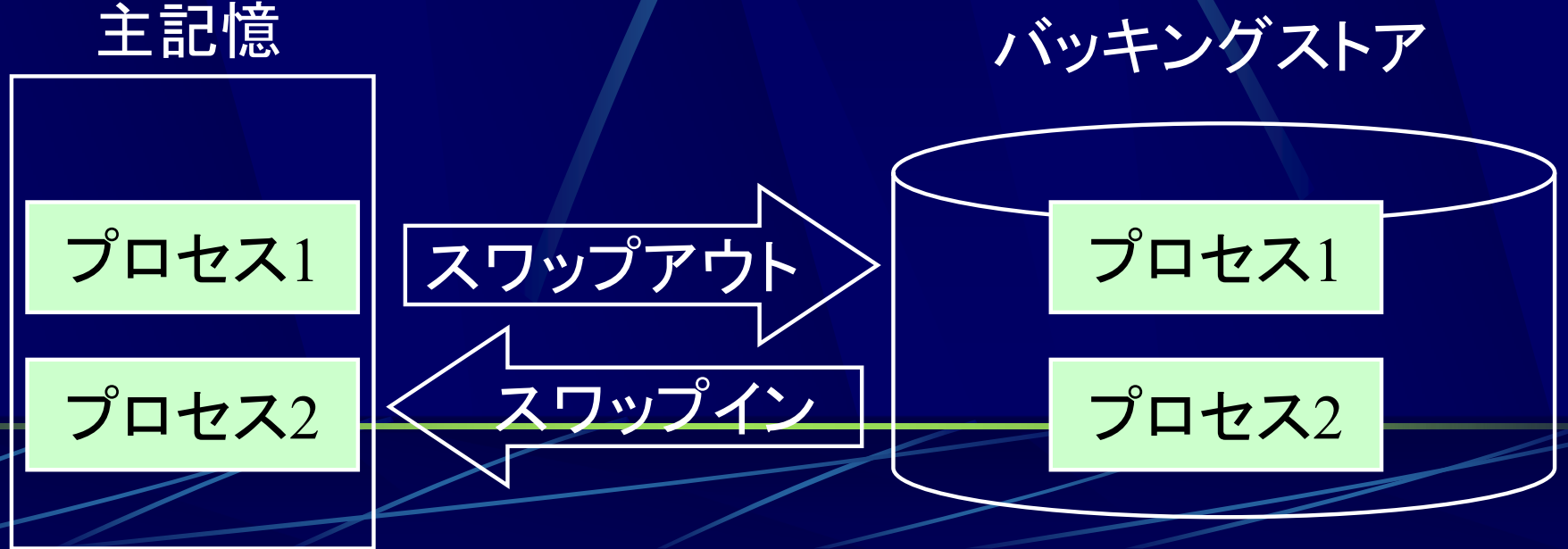


プロセッサは2次記憶を直接読むことはできない

使用するプログラム，データは主記憶上にコピー

スワッピング(swapping)

- スワッピング(swapping)
  - 待ち状態のプロセスを2次記憶に退避させる
- バッキングストア(backing store)
  - スワッピングを行う際に使用する2次記憶

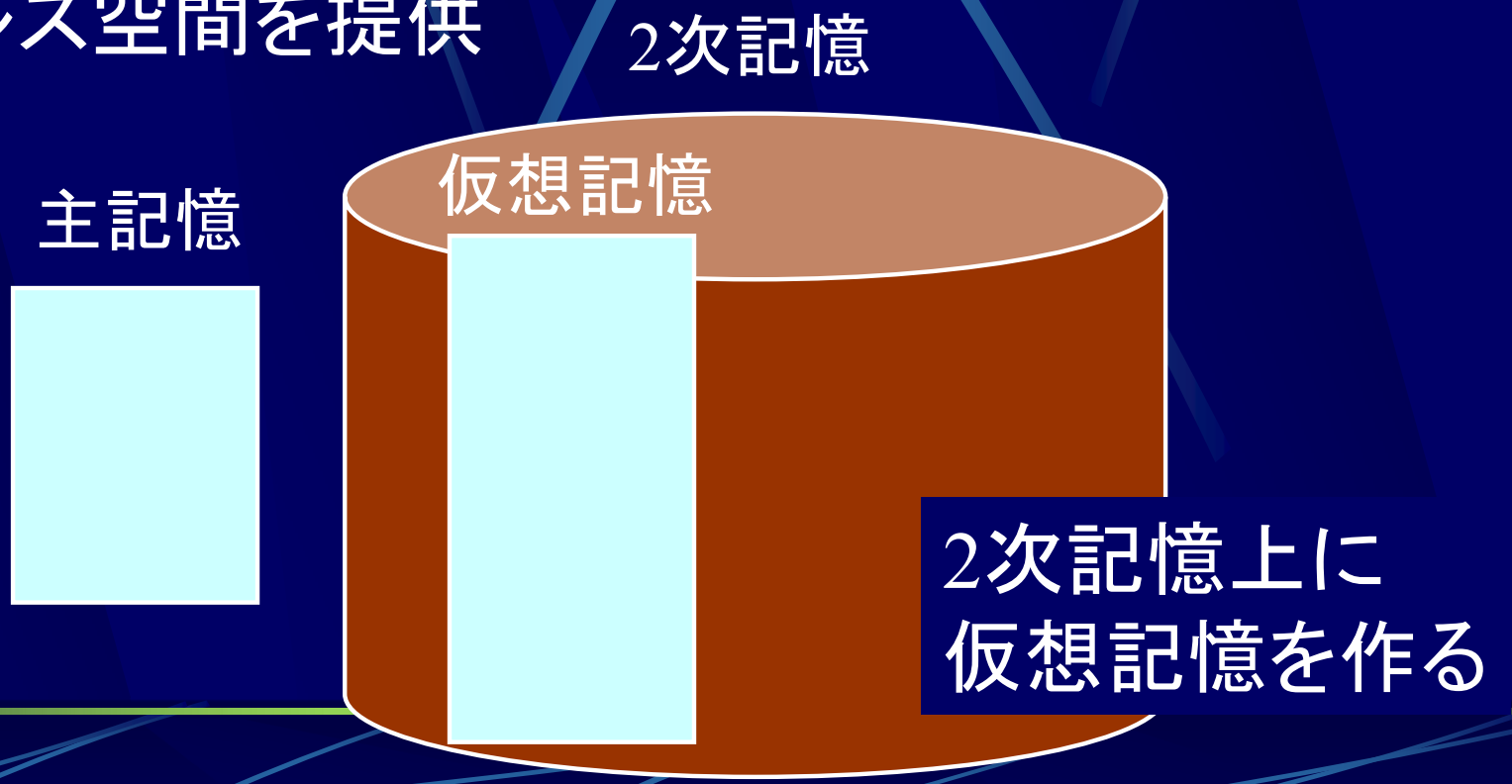


スワップイン, スワップアウト
(swap-in, swap-out)

- スワップイン(swap-in)
  - プログラム, データを2次記憶から主記憶に
  - 実行中のプロセスに必要なものを読み込む
- スワップアウト(swap-out)
  - プログラム, データを主記憶から2次記憶に
  - スワップインの領域を確保するために当面必要のないものを退避させる

仮想記憶(virtual memory)

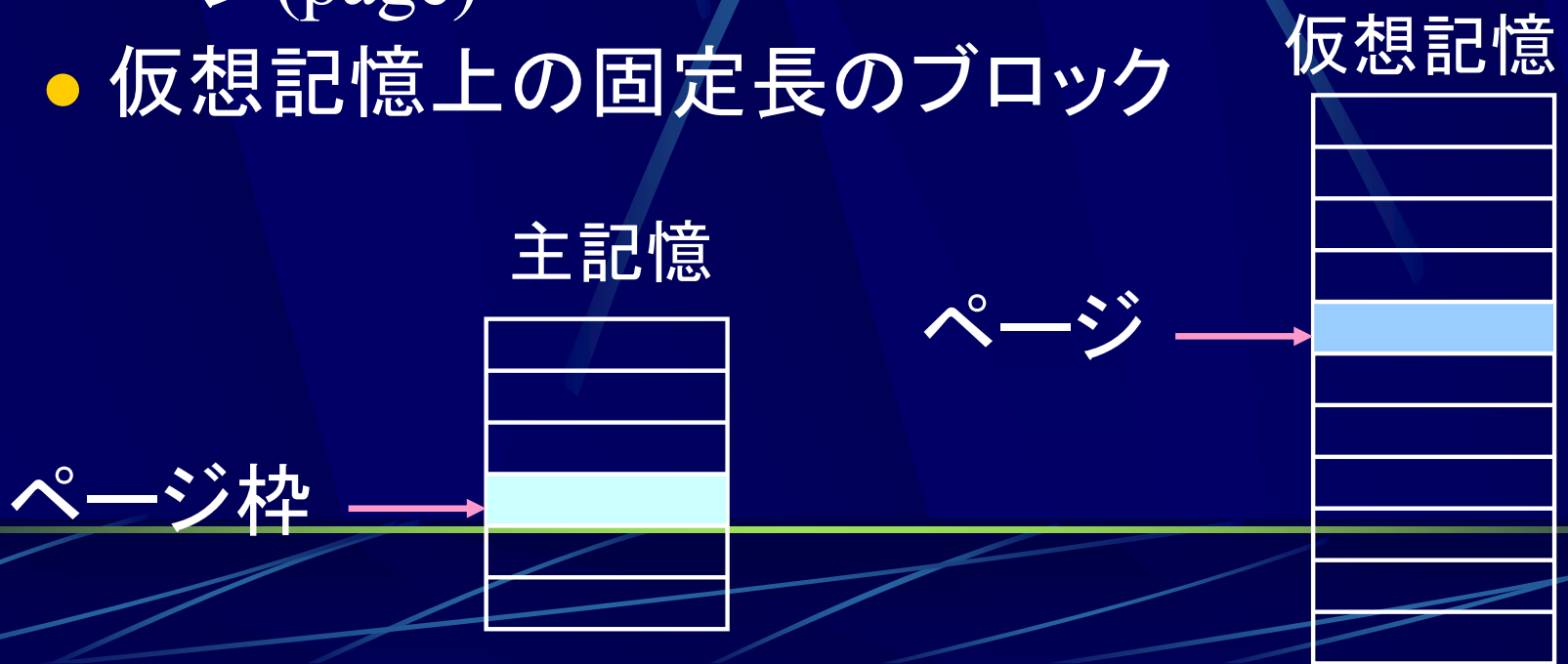
- 仮想記憶(virtual memory)
  - 動的再配置により主記憶容量よりも大きなアドレス空間を提供



ページング(paging)

- ページング(paging)
  - 主記憶, 仮想記憶を固定長のブロックに分割
- ページ枠(page frame)
  - 主記憶上の固定長のブロック
- ページ(page)
  - 仮想記憶上の固定長のブロック

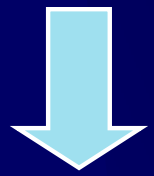
サイズ 2^k KB
4～8KB



ページング

必要なプログラム, データの載っているページが

1. 主記憶上にある場合



そのまま実行

ページング

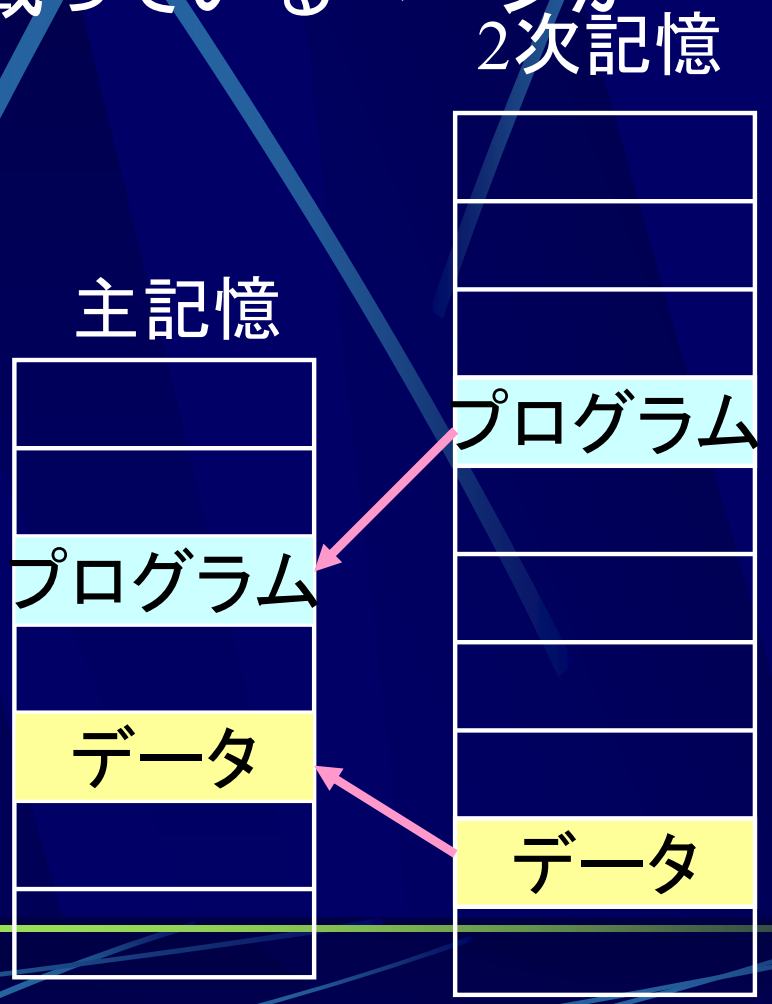
必要なプログラム, データの載っているページが

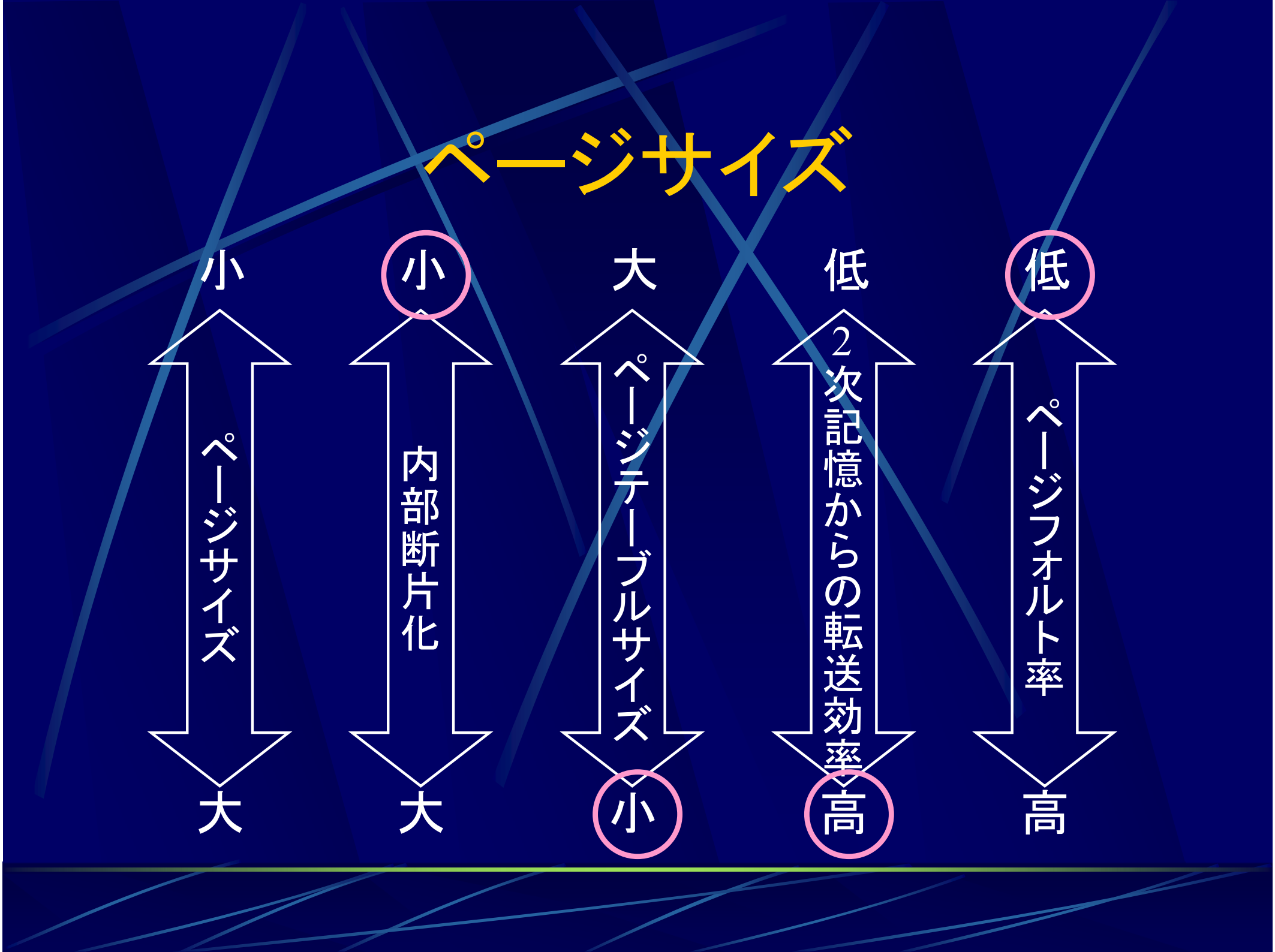
2. 主記憶上に無い場合

ページフォルト
(page fault)

2次記憶から主記憶に
ページを読み込む

ページイン(page in)




ページサイズ
小
ページサイズ
大
小
内部断片化
大
大
ページテーブルサイズ
小
低
2次記憶からの転送効率
高
低
ページフォルト率
高

ページングの問題点

- ページテーブルが巨大

  例: 仮想記憶を 4GB, 1ページ 8KB で構成

  ページエントリ数 約50万

- ページテーブルはプロセスごとに独立

  例: 100個のプロセスを実行

  ページエントリ数 約5000万

  1エントリ10Bなら 約500MB

ページングの問題点の解法

- ページテーブルが巨大
  - ハッシュ(hash)関数によるページテーブル

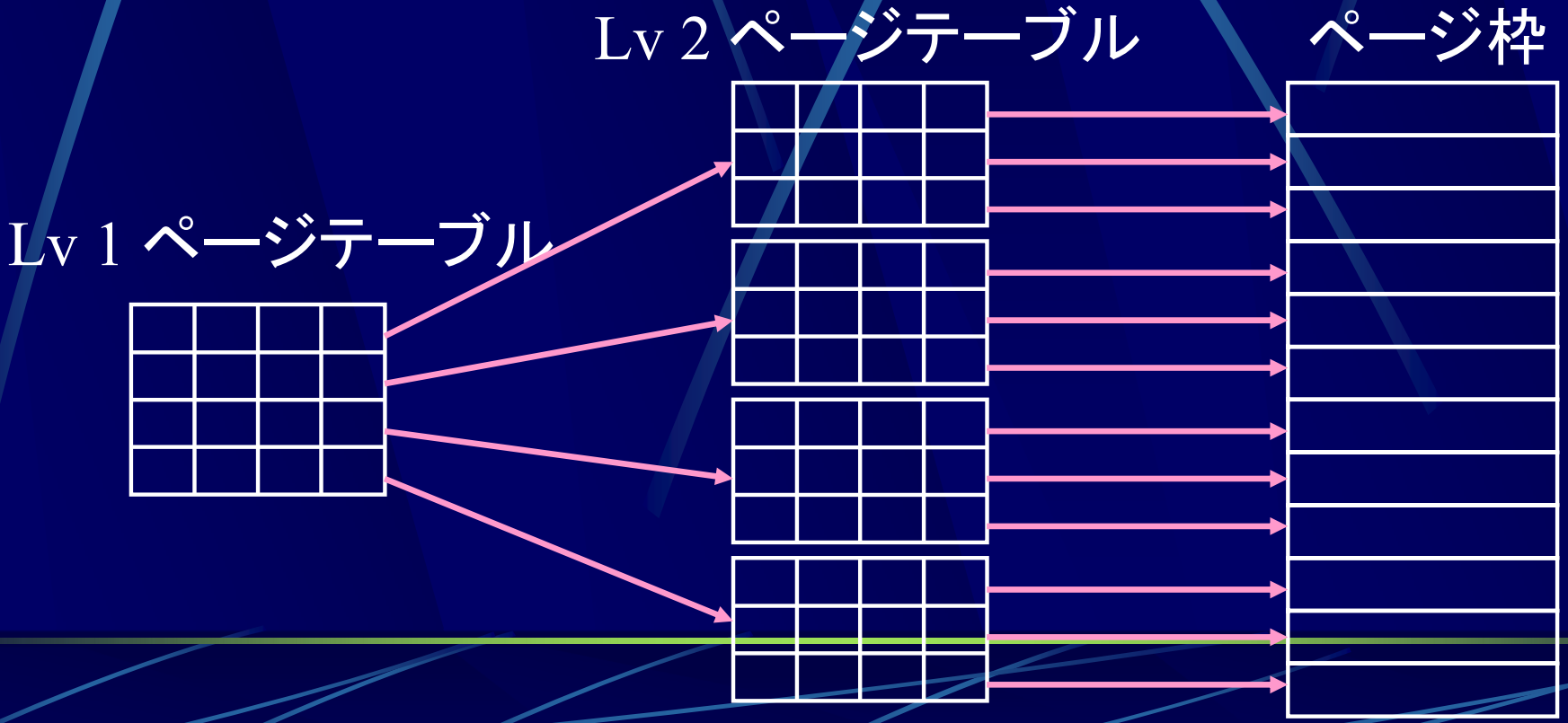
ページ数(仮想記憶) ⇒ ページ枠数(実記憶)

もっと改善するには?

多重レベルページング

多重レベルページング (multilevel paging)

- 多重レベルページング(multilevel paging)
  - ページングを多段化



多重レベルページング

Lv1 ページテーブル

ページ	ページ枠	フラグ
00	4	1
01	6	1
02	2	1

主記憶

ページ枠	ページ
0	
1	
2	
3	
4	
5	
6	
7	

ページ02の
Lv2ページテーブル

ページ00の
Lv2ページテーブル

ページ01の
Lv2ページテーブル

ページ枠に
Lv2ページングの
ページテーブル

多重レベルページング

仮想アドレスの構成

<table>
<tr><td colspan="3">仮想アドレス</td></tr>
<tr><td rowspan="2">ページ番号
Lv 1</td><td colspan="2">Lv1ページ内相対アドレス</td></tr>
<tr><td>ページ番号
Lv 2</td><td>Lv2ページ内相対アドレス</td></tr>
</table>

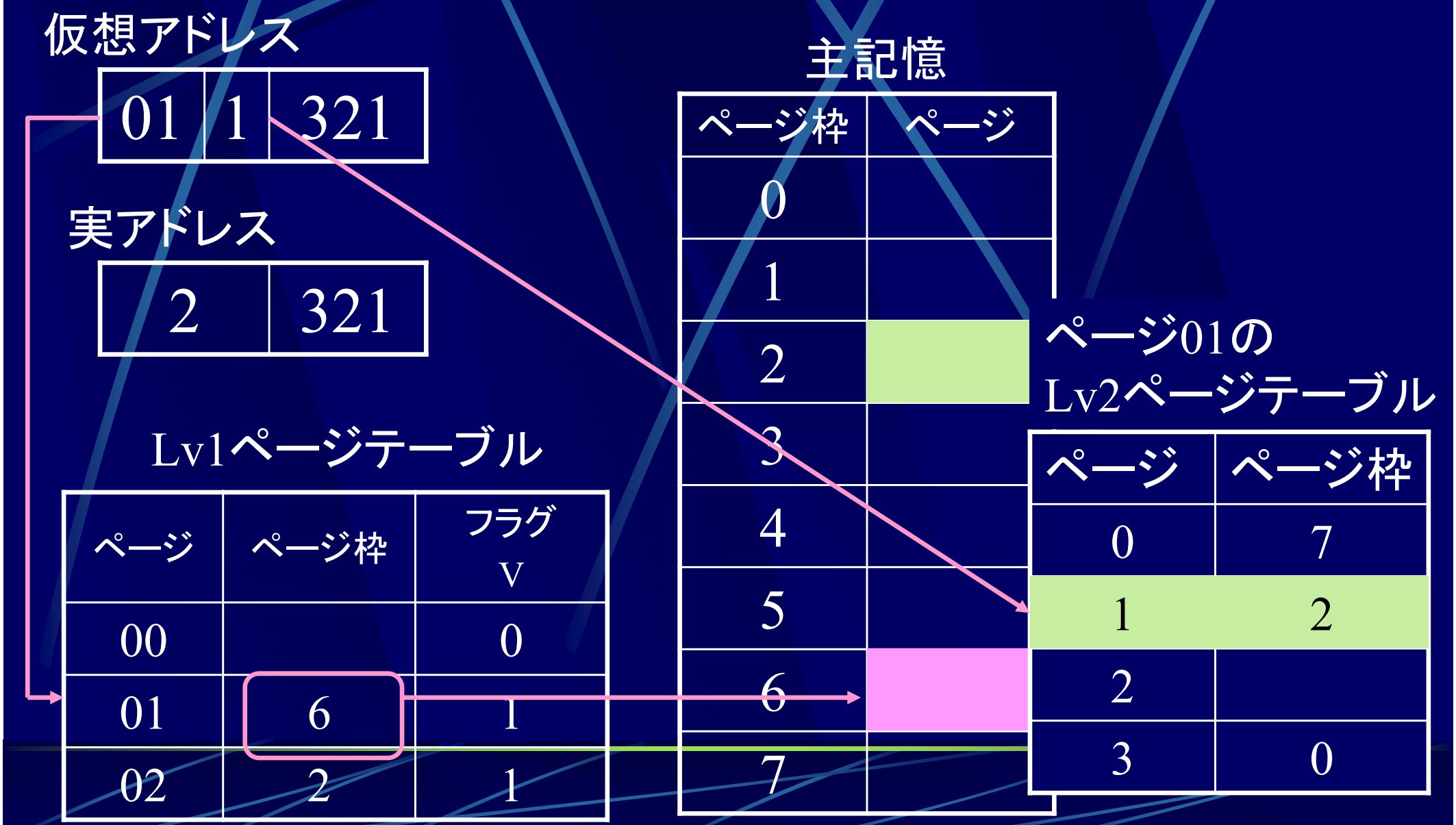
アドレスが二重構造

例

01	1	321

Lv1ページ内相対アドレス
＝Lv2ページングの仮想アドレス

多重レベルページング


仮想アドレス
01
1
321
実アドレス
2
321
Lv1ページテーブル
ページ
ページ枠
フラグ V
00
01
6
1
02
2
1
主記憶
ページ枠
ページ
0
1
2
3
4
5
6
7
ページ01の Lv2ページテーブル
ページ
ページ枠
0
7
1
2
2
3
0

多重レベルページング

仮想アドレス

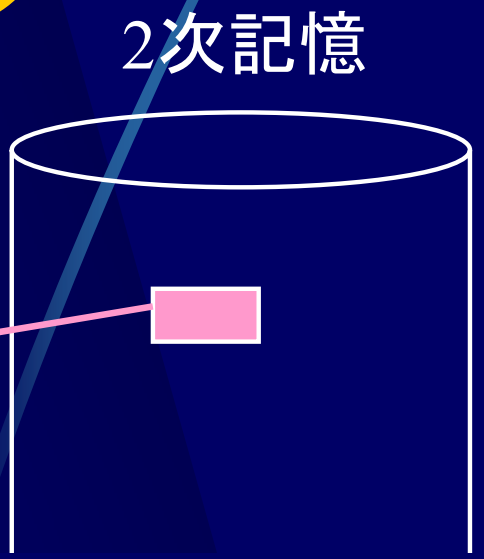
00	3	555

Lv1ページテーブル

ページ	ページ枠	フラグ V
00		0
01	6	1
02	2	1

主記憶

ページ枠	ページ
0	
1	
2	
3	
4	
5	
7	



2次記憶から
Lv2ページテーブルを
ページイン

ページフォルト発生

多重レベルページング

仮想アドレス

00	3	555

実アドレス

4	555

Lv1ページテーブル

ページ	ページ枠	フラグ V
00	0	1
01	6	1
02	2	1

主記憶

ページ枠	ページ
0	
1	
2	
3	
4	
5	
6	
7	

ページ00の
Lv2ページテーブル

ページ	ページ枠
0	
1	7
2	6
3	4

多重レベルページングの利点

- 多重レベルページングの利点
  - 主記憶使用量の削減
    - 当面必要ではないページテーブルは2次記憶に置く
- 多重レベルページングの欠点
  - メモリアクセスの増大
    - 連想レジスタを用いればアクセス数を減らせる

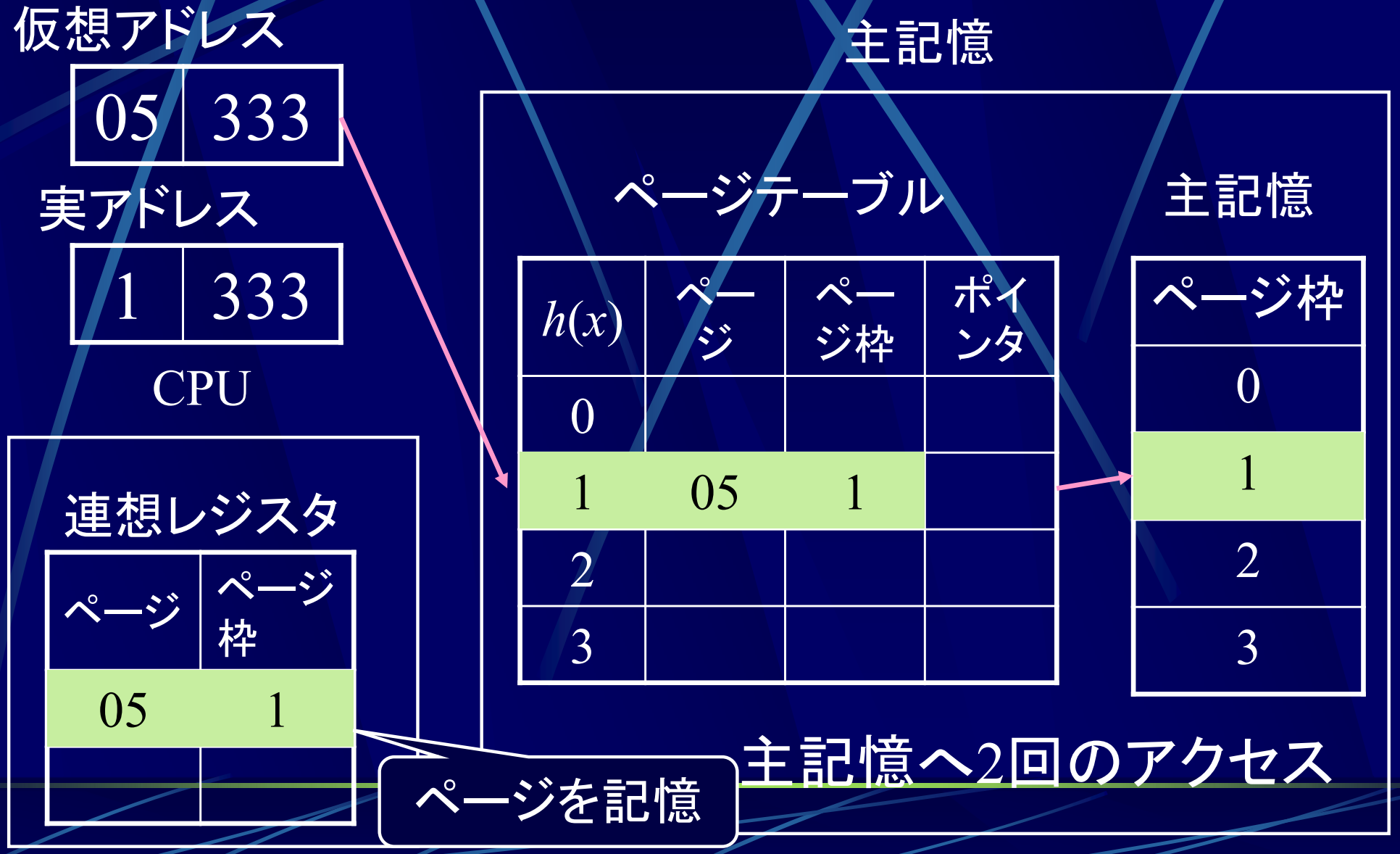
連想レジスタ

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
再アクセスすることが多い

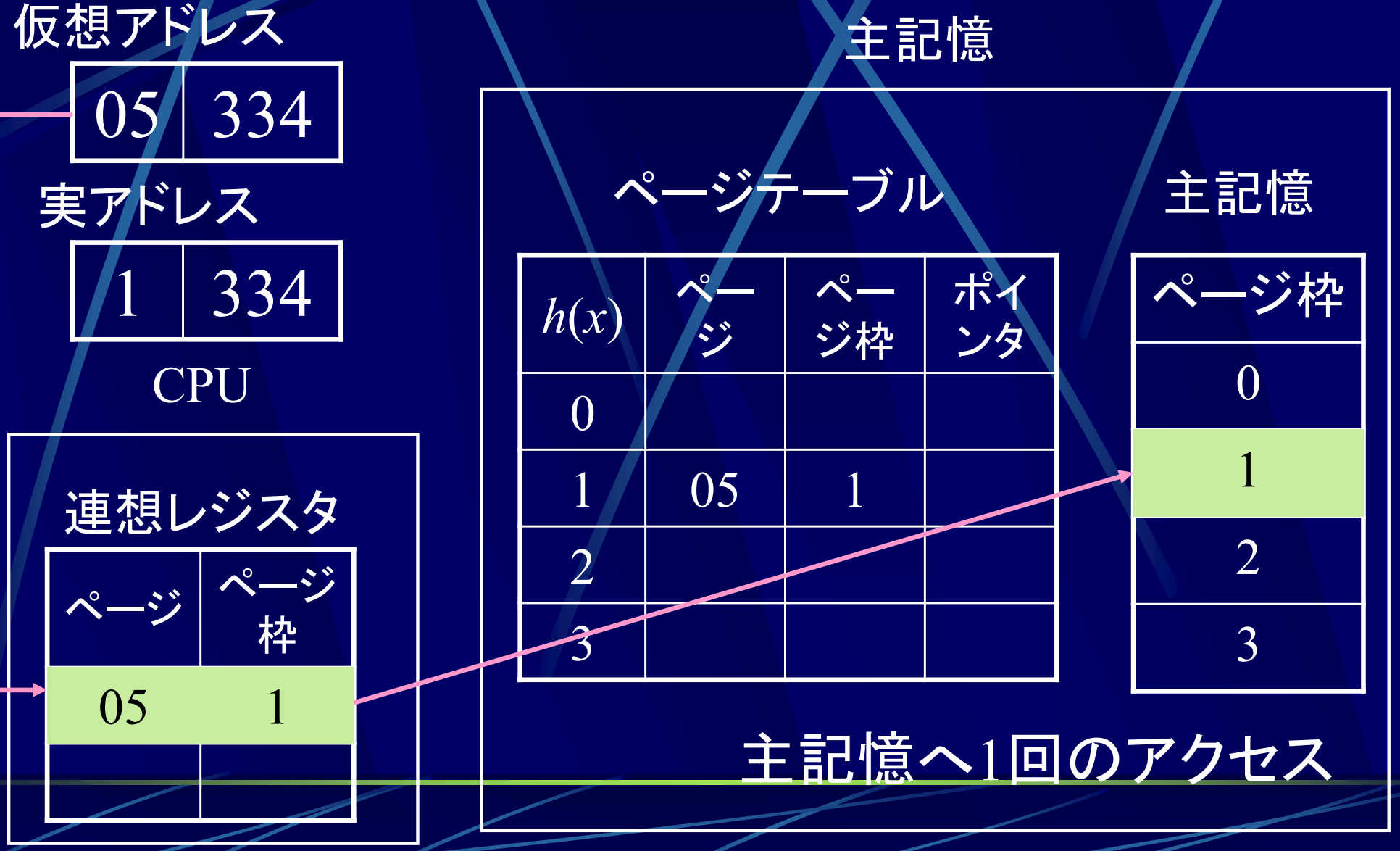
最近参照されたページテーブルをCPU内で記憶

連想レジスタ
- 小容量
- 高速

連想レジスタ


仮想アドレス
05
333
実アドレス
1
333
CPU
連想レジスタ
ページ
ページ枠
05
1
主記憶
ページテーブル
h(x)
ページ
ページ枠
ポインタ
0
1
05
1
2
3
主記憶
ページ枠
0
1
2
3
ページを記憶
主記憶へ2回のアクセス

連想レジスタ


仮想アドレス
05
334
実アドレス
1
334
CPU
連想レジスタ
ページ
ページ枠
05
1
主記憶
ページテーブル
h(x)
ページ
ページ枠
ポインタ
0
1
05
1
2
3
主記憶
ページ枠
0
1
2
3
主記憶へ1回のアクセス

連想レジスタ

- 主記憶へのアクセス回数 (通常のページング)
  - 初めてアクセスするページ　　：2回
  - 最近アクセスしたページ　　：1回
- 主記憶へのアクセス回数 (多重レベルページング)
  - 初めてアクセスするページ　　：3回
  - 最近アクセスしたページ　　：1回

統計的にはレジスタが8～16 個で
90%のヒット率

理想的な仮想記憶

ページングで可能

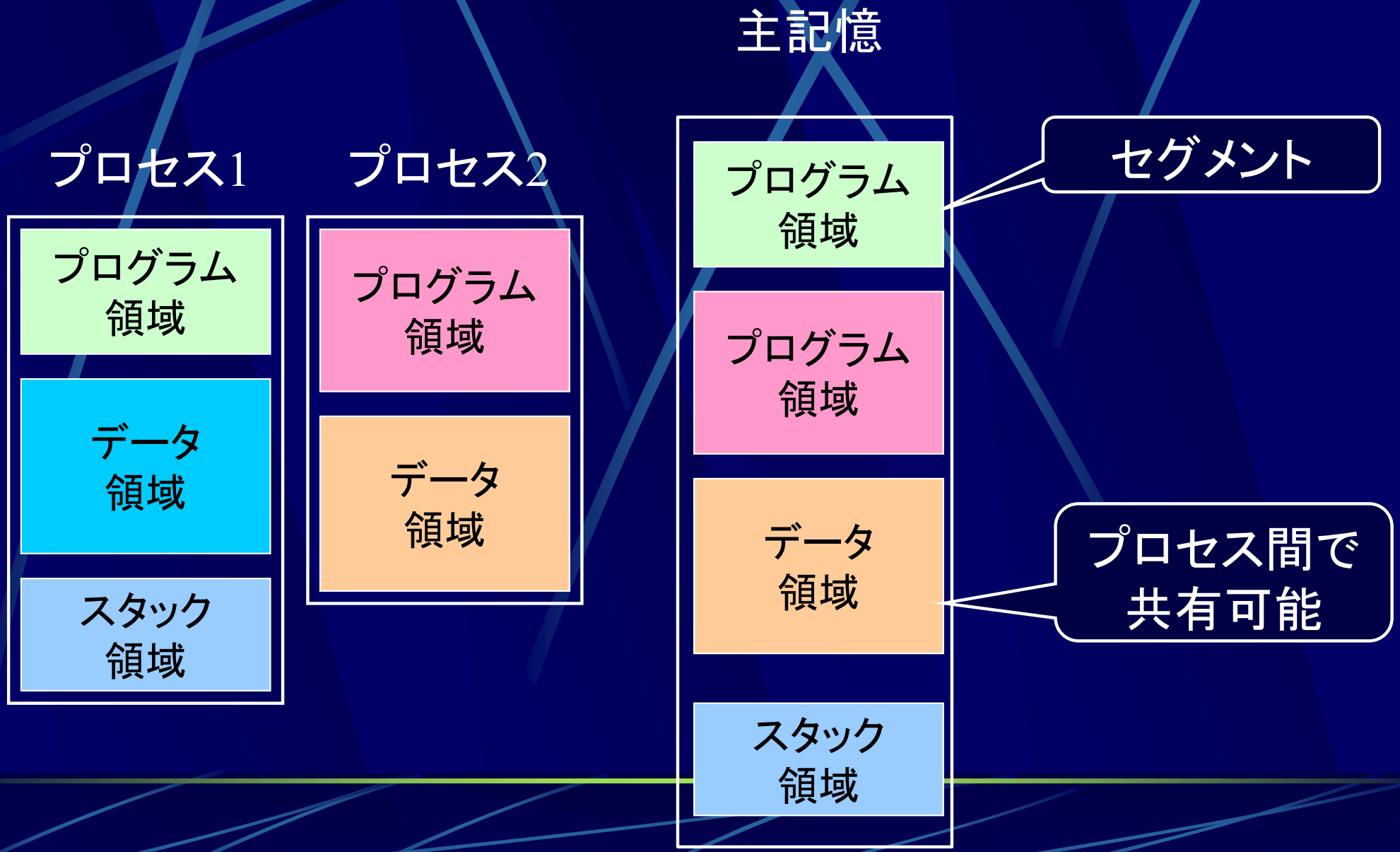
- 大きさは無制限
  - プロセスは記憶容量を考慮する必要無し
  - プログラムの簡単化, バグの可能性減少
- プロセス毎に固有
  - 他プロセスからのアクセスに対し保護
- プログラム部, データ部, スタック部等を分離
  - プロセス内部からのアクセスに対し保護
- 必要時にはプロセス間で共有可能
  - 並列動作するプロセス間で通信機構として使用

これも実現したい

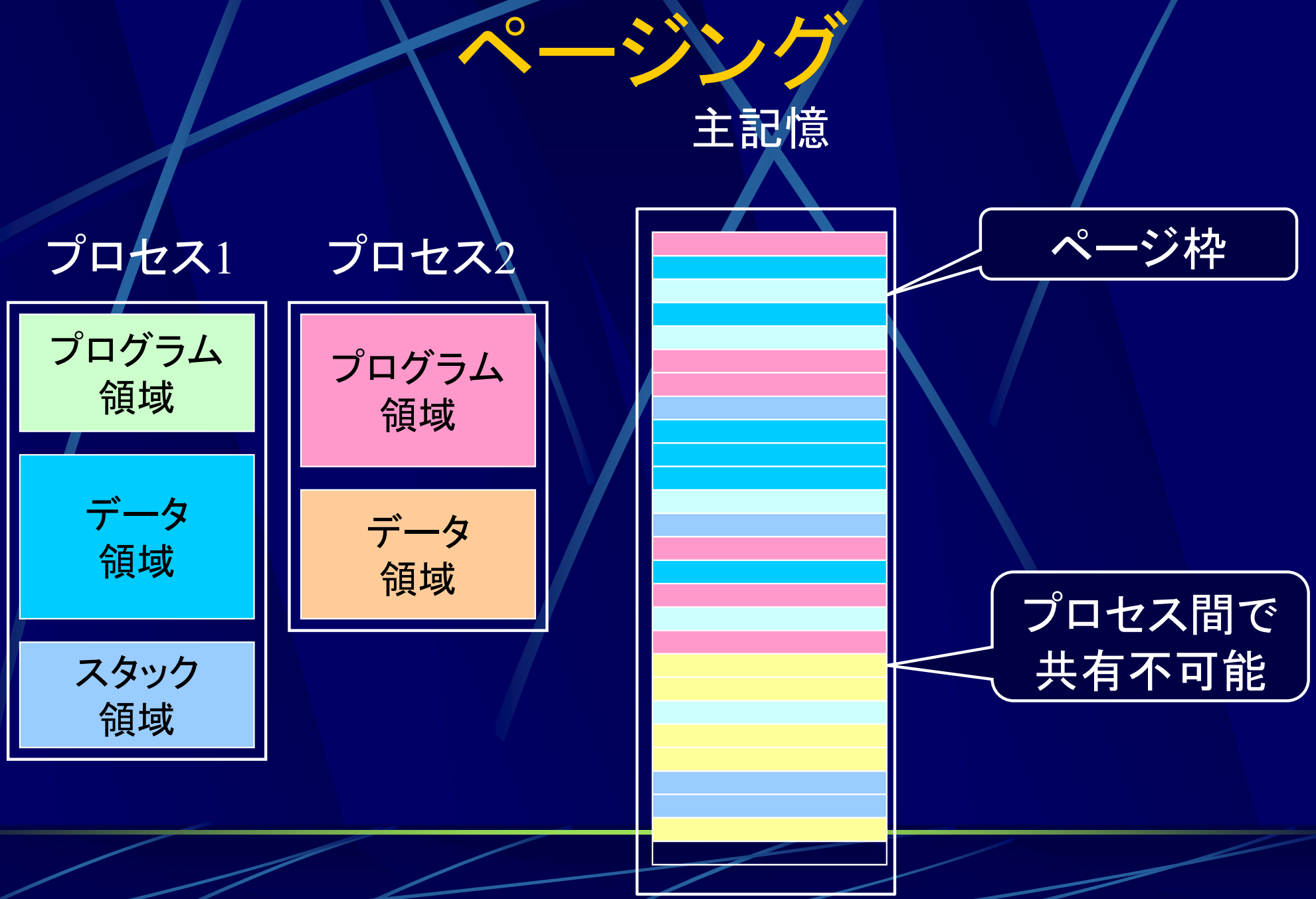
セグメンテーション (segmentation)

- セグメント(segment)
  - プログラム, データの論理的なかたまり
    - メインルーチン, サブルーチン, データ等
- セグメンテーション(segmentation)
  - セグメントを単位として分割
  - 各セグメントは大きさを自由に増減可能

セグメンテーション


主記憶
プロセス1
プログラム領域
データ領域
スタック領域
プロセス2
プログラム領域
データ領域
プログラム領域
プログラム領域
データ領域
スタック領域
セグメント
プロセス間で共有可能

ページング


主記憶
プロセス1
プログラム領域
データ領域
スタック領域
プロセス2
プログラム領域
データ領域
ページ枠
プロセス間で共有不可能

セグメンテーション

必要なプログラム, データの載っているセグメントが

1. 主記憶上にある場合

そのまま実行

セグメンテーション

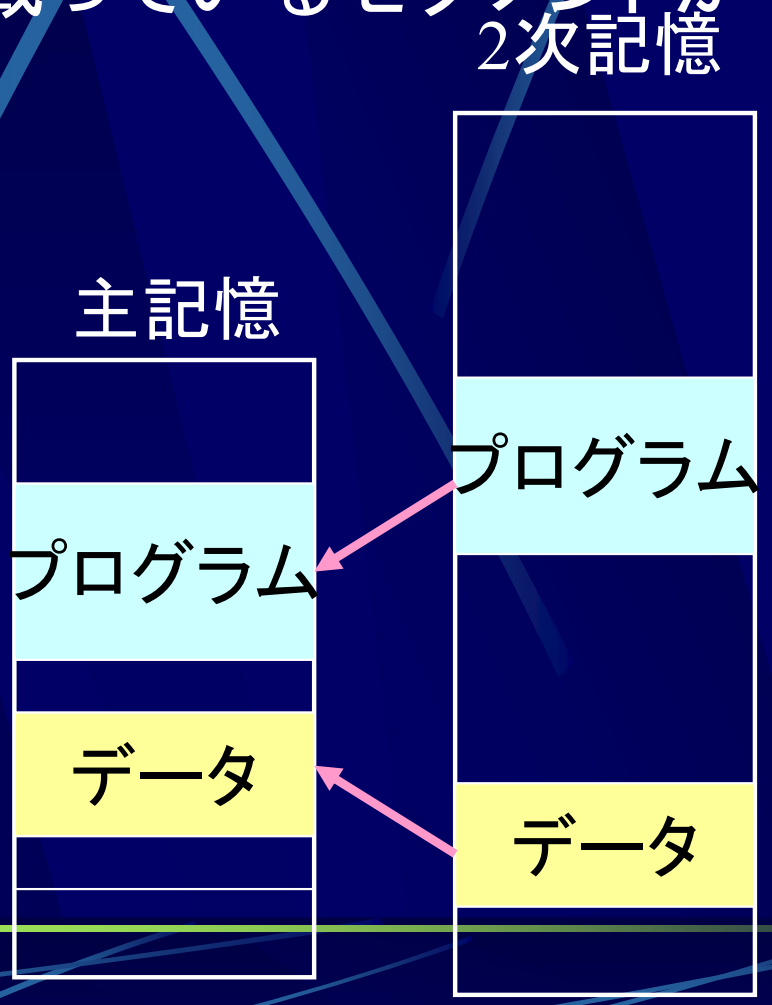
必要なプログラム, データの載っているセグメントが

2. 主記憶上に無い場合

セグメントフォルト
(segment fault)

2次記憶から主記憶に
セグメントを読み込む

セグメントイン(segment in)



セグメントテーブル(segment table)

- セグメントテーブル(segment table)
  - 仮想アドレスから実アドレスへの変換表

セグメント番号	基底アドレス	セグメント長	フラグ		
			V	P	C
00	2250	550	1	100	1
01		300	0	110	0
02	1100	1200	1	101	1
03		850	0	111	0
04		600	0	110	0
05		150	0	111	0

セグメントテーブル(segment table)

- セグメントテーブル(segment table)
  - 仮想アドレスから実アドレスへの変換表

セグメント番号	基底アドレス	セグメント長	フラグ		
			V	P	C
00	2250	550	1	100	1
01		300	0	110	0
02			1	101	1
03		850	0	111	0
04		600	0	110	0
05		150	0	111	0

仮想記憶上のセグメント番号

セグメントテーブル(segment table)

- セグメントテーブル(segment table)
  - 仮想アドレスから実アドレスへの変換表

セグメント番号	基底アドレス	セグメント長	フラグ		
			V	P	C
00		550	1	100	1
01		300	0	110	0
02				101	1
03		850	0	111	0
04		600	0	110	0
05		150	0	111	0

主記憶上のセグメントの開始位置

セグメントテーブル(segment table)

- セグメントテーブル(segment table)
  - 仮想アドレスから実アドレスへの変換表

セグメント番号	基底アドレス	セグメント長	フラグ		
			V	P	C
00	2250	5[illegible]0	1	100	1
01			0	110	0
02	1100		1	101	1
03		850	0	111	0
04		600	0	110	0
05		150	0	111	0

セグメントの長さ

セグメントテーブル(segment table)

- セグメントテーブル(segment table)
  - 仮想アドレスから実アドレスへの変換表

セグメント番号	基底アドレス	セグメント長	フラグ		
			V	P	C
00	2250	550	1	100	1
01		300	0	110	0
02				01	1
03				11	0
04		600	0	110	0
05		150	0	111	0

Virtual Memory Flag
実記憶上にスワップインされているか

セグメントテーブル(segment table)

- セグメントテーブル(segment table)
  - 仮想アドレスから実アドレスへの変換表

セグメント番号	基底アドレス	セグメント長	フラグ		
			V	P	C
00	2250	550	1	100	1
01		300		110	0
02	110				1
03					0
04		600	0	110	0
05		150	0	111	0

Permission Flag
アクセス権 (read, write, execute)

セグメントテーブル(segment table)

- セグメントテーブル(segment table)
  - 仮想アドレスから実アドレスへの変換表

セグメント番号	基底アドレス	セグメント長	フラグ		
			V	P	C
00	2250	550	1	100	1
01		300	0		0
02	1100				
03					
04		600	0	110	0
05		150	0	111	0

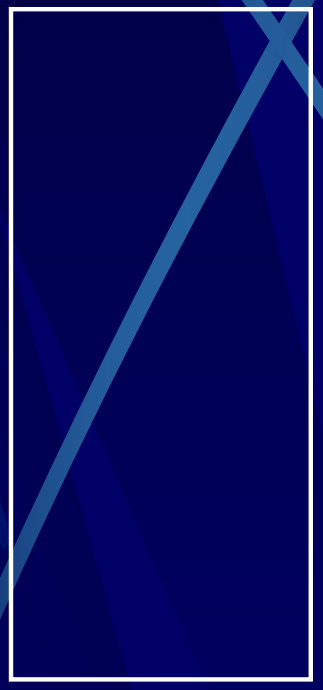
Change Flag
セグメントイン後に書き換えられたか

セグメンテーション

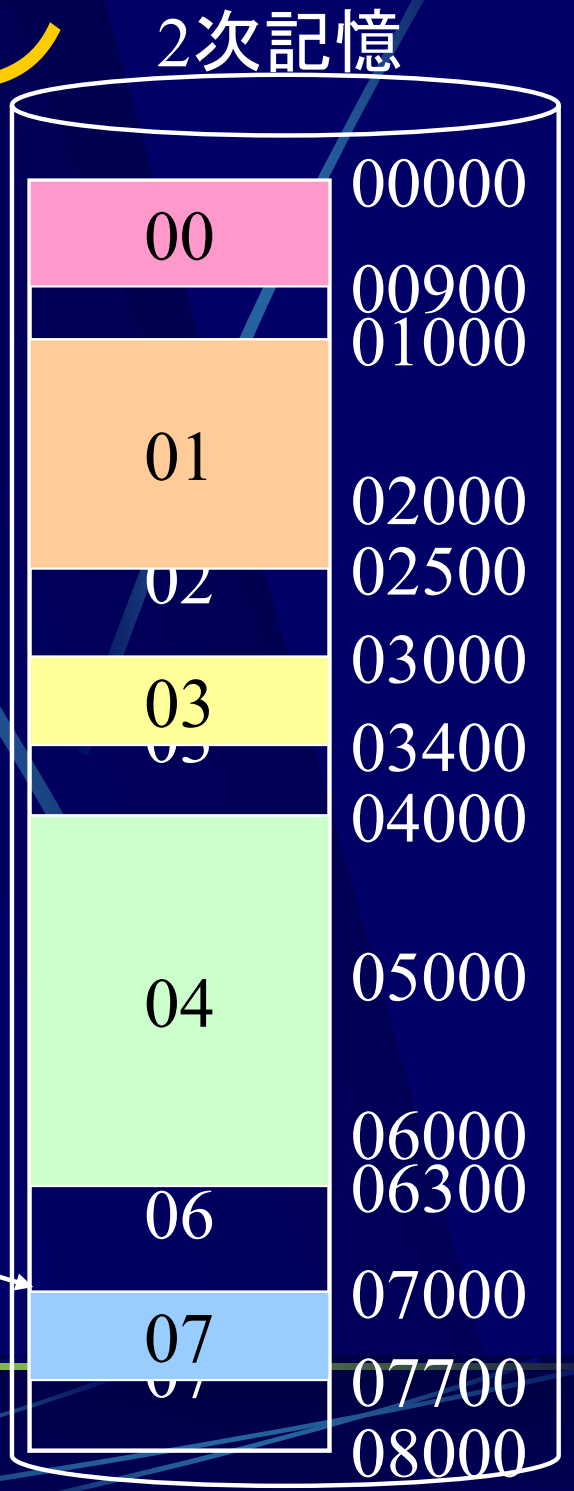
セグメントテーブル

番号	基底アドレス	セグメント長	フラグ V
00		900	0
01	1200	1500	1
02		-	0
03	3100	400	1
04		2300	0
05		-	0
06		-	0
07		700	0

主記憶



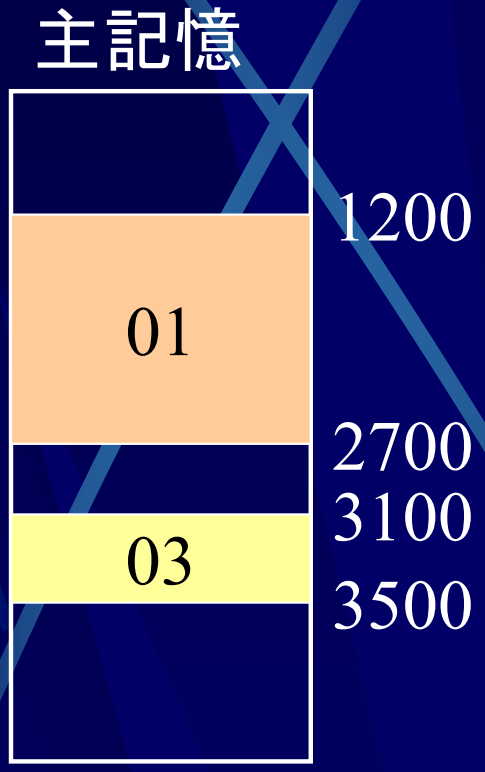
セグメントの開始位置



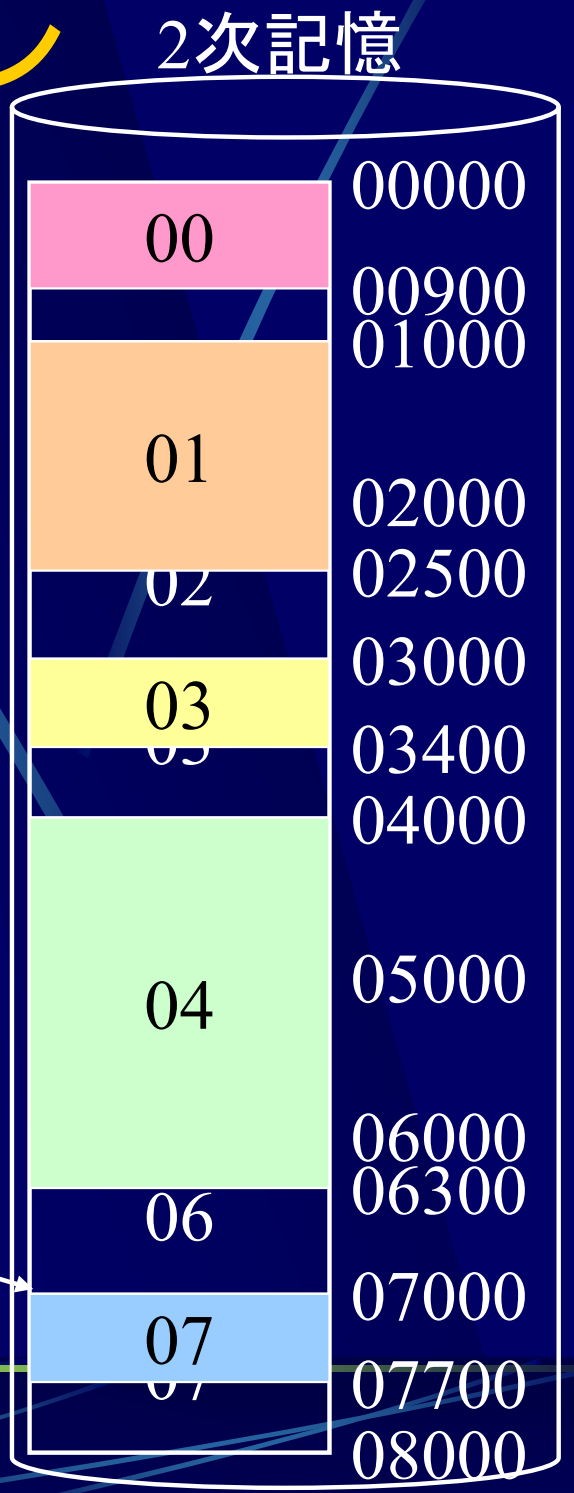
セグメンテーション

セグメントテーブル

番号	基底アドレス	セグメント長	フラグ V
00		900	0
01	1200	1500	1
02		-	0
03	3100	400	1
04		2300	0
05		-	0
06		-	0
07		700	0



セグメントの開始位置



セグメンテーション

主記憶上にある場合

仮想アドレス

01	321

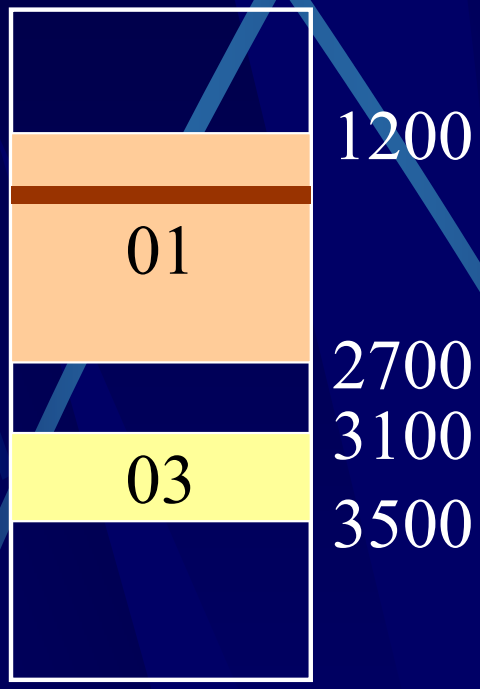
実アドレス

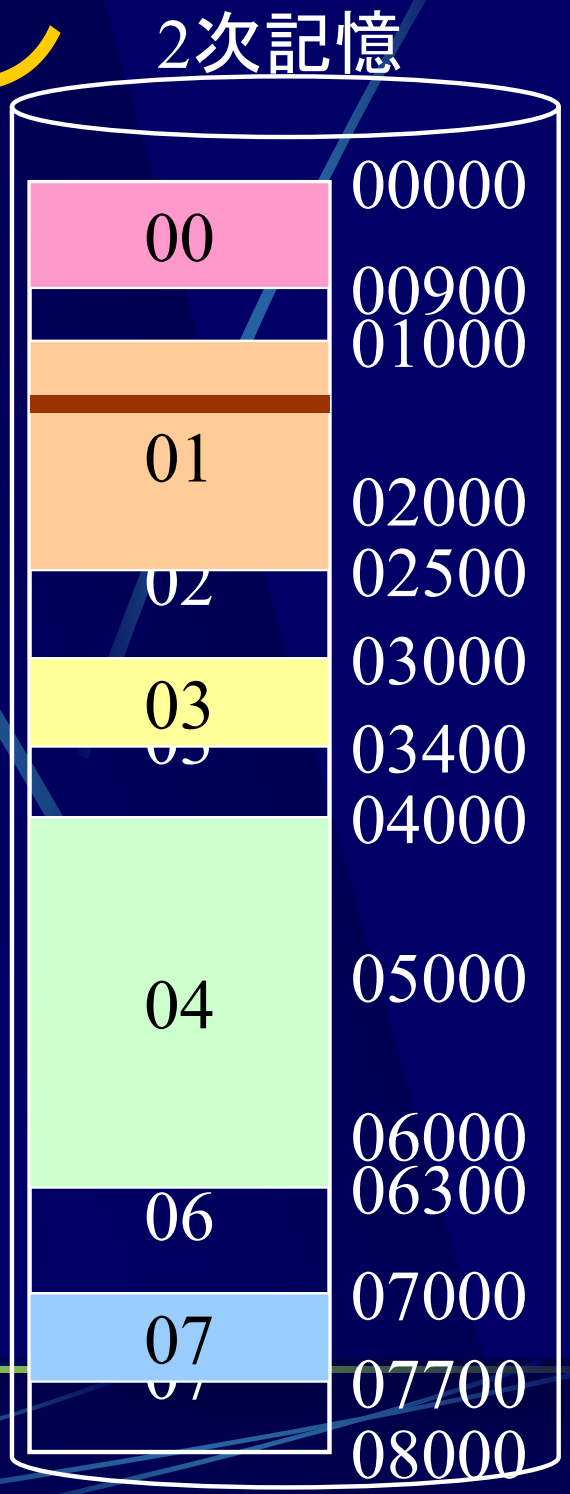
1521

1200 + 321

セグメントテーブル

番号	基底アドレス	セグメント長	フラグ V
00		900	0
01	1200	1500	1
02		-	0
03	3100	400	1





セグメンテーション

主記憶上に無い場合

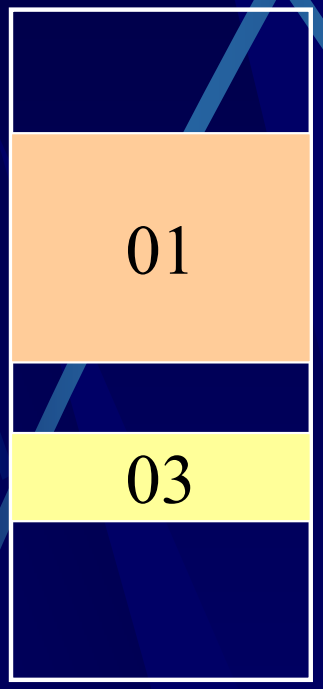
仮想アドレス

00	888

セグメントテーブル

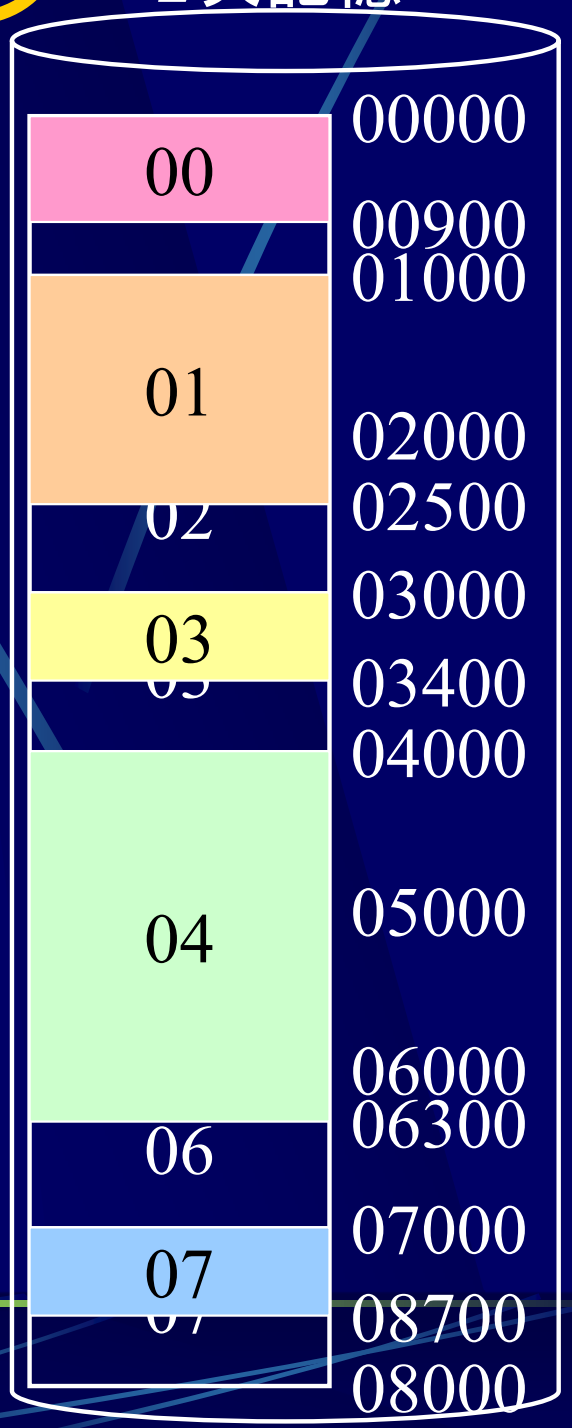
番号	基底アドレス	セグメント長	フラグ V
00		900	0
01	1200	1500	1
02		-	0
03	3100	400	1

主記憶



セグメント
フォルト発生

2次記憶



セグメンテーション

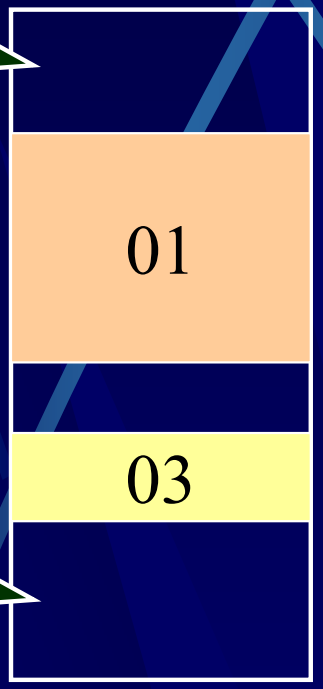
主記憶上に無い場合

仮想アドレス

00	888

主記憶

1200



1200
2700
3100
3500

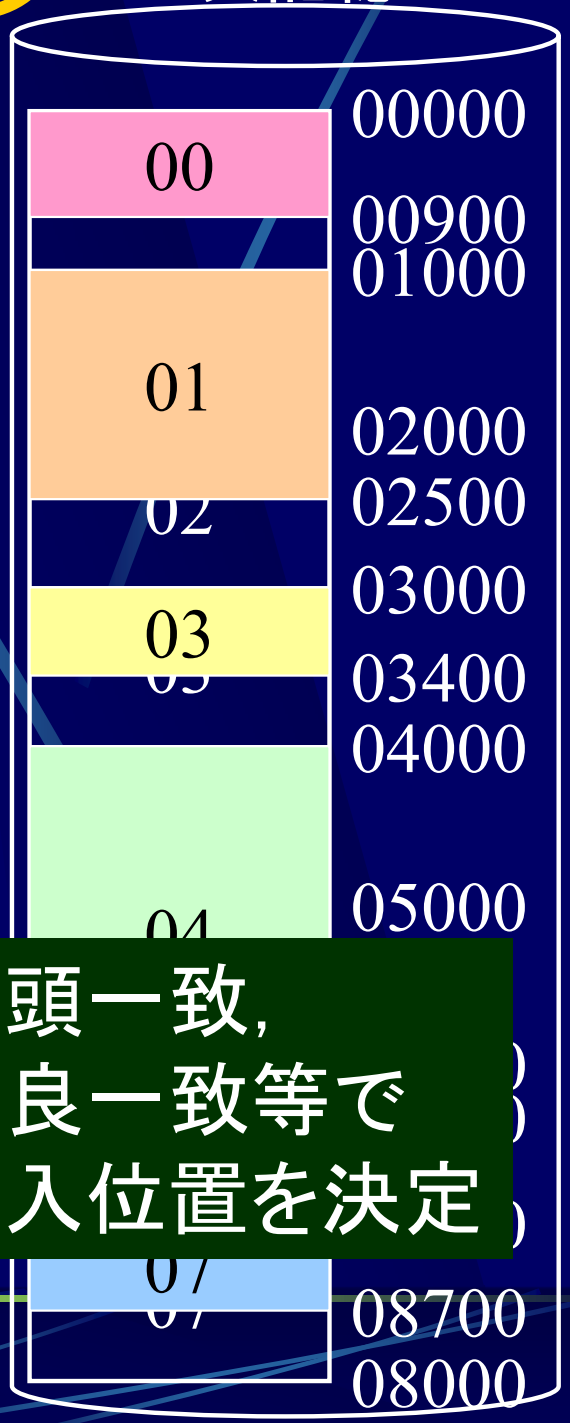
セグメントテーブル

1000

番号	基底アドレス	セグメント長	V
00		900	0
01	1200	150	
02		-	
03	3100	400	1

セグメント長以上の空き領域を探す

2次記憶



00000
00900
01000
02000
02500
03000
03400
04000
05000
08700
08000

先頭一致，
最良一致等で
挿入位置を決定

セグメンテーション

主記憶上に無い場合

仮想アドレス

00	888

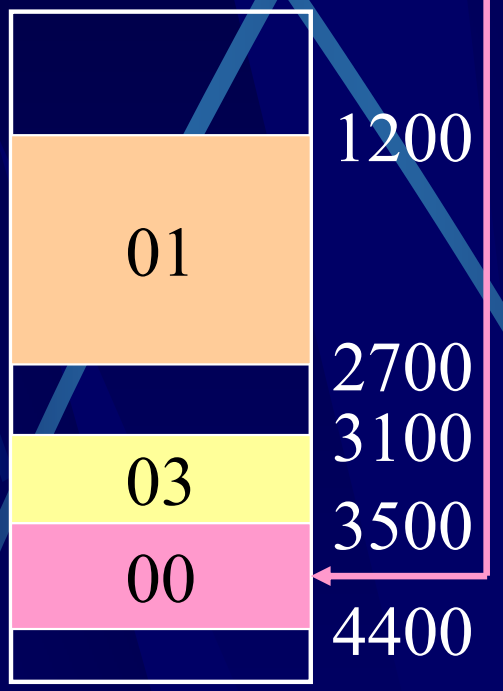
実アドレス

4388

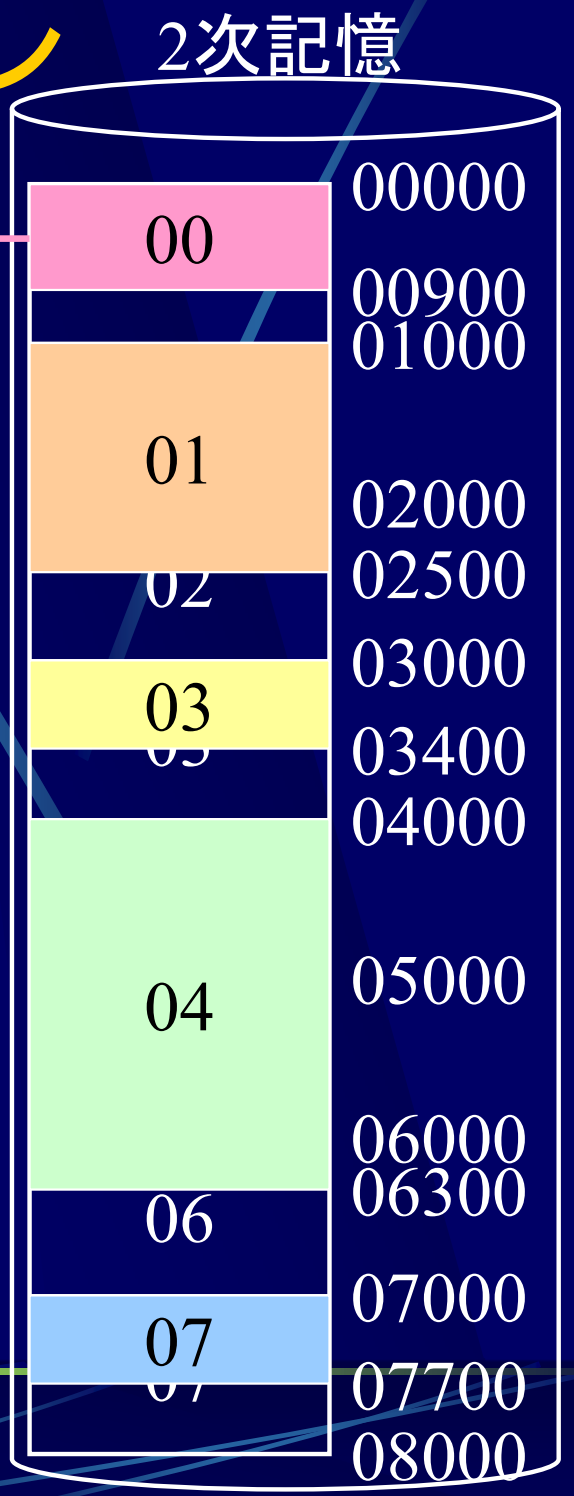
3500 + 888

セグメントテーブル

番号	基底アドレス	セグメント長	フラグ V
00	3500	900	1
01	1200	1500	1
02		-	0
03	3100	400	1



セグメントイン



セグメンテーション

仮想アドレス

03	550

セグメント内相対アドレス > セグメント長

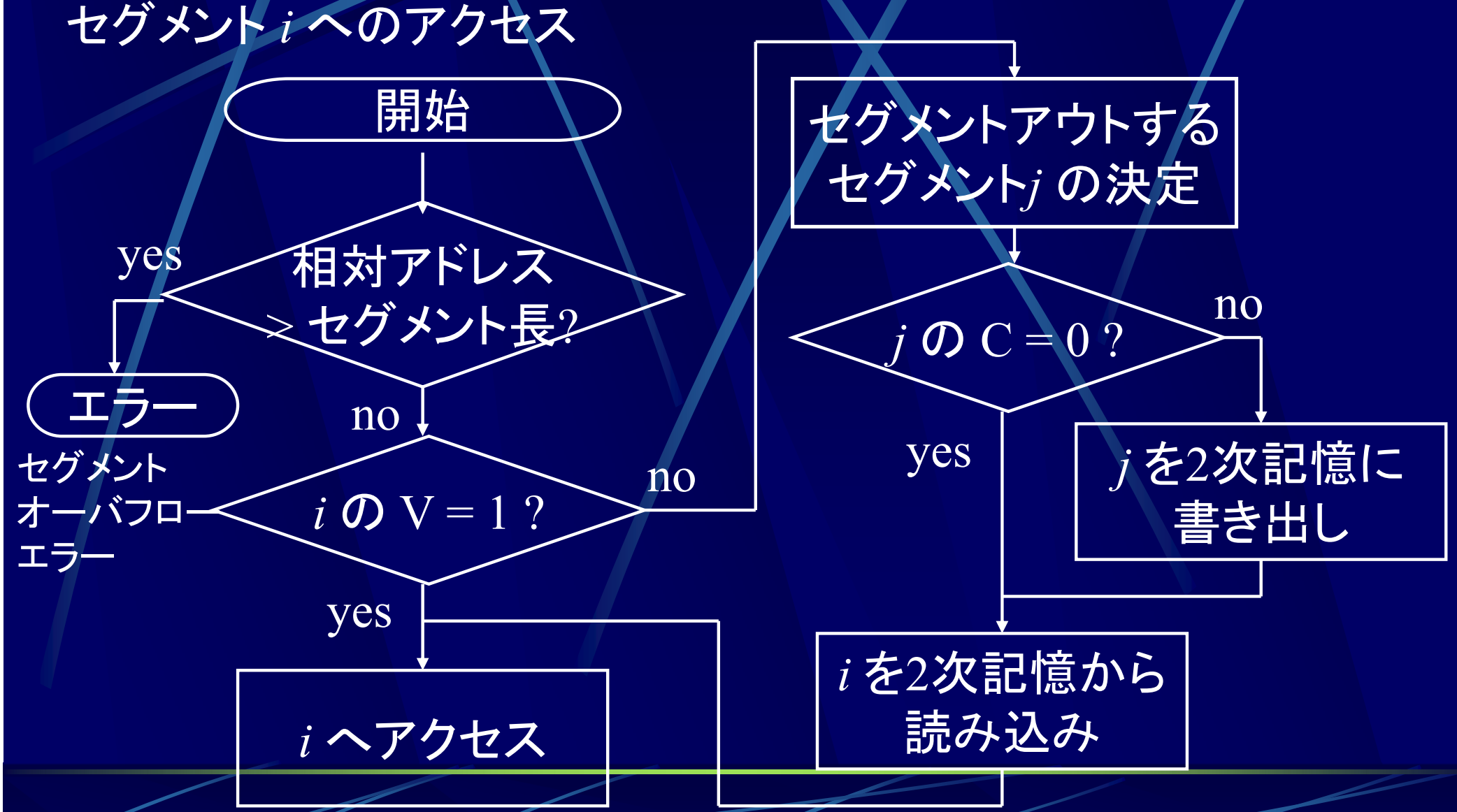
セグメントオーバフローエラー

(segment overflow error)

セグメントテーブル

番号	基底アドレス	セグメント長	フラグ V
00	3500	900	V
01	1200	1500	1
02		-	0
03	3100	400	1

セグメンテーションの動作



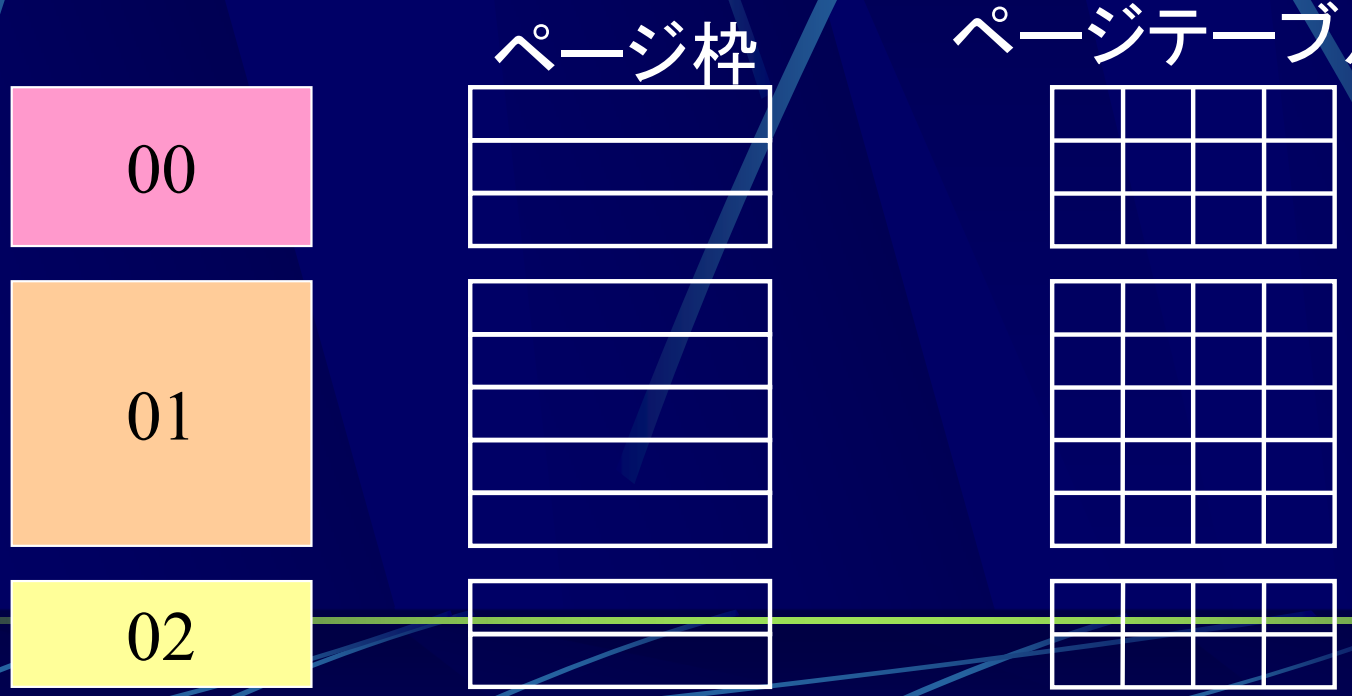
セグメンテーションの長所と短所

- 長所
  - 大きさを増減可能
  - プログラム部, データ部等用途別に割り当て可能
- 短所
  - 外部断片化が発生

ページングと組み合わせる

ページ化セグメンテーション
(paged segmentation)

- ページ化セグメンテーション(paged segmentation)
  - セグメントを複数のページ枠で構成
  - セグメントごとにページテーブルを用意



ページ化セグメンテーション

セグメントテーブル

番号	基底アドレス	セグメント長	フラグ
00	4,000	900	1
01	6,000	1500	1
02	1,000	500	1

ページ枠に
各セグメントの
ページテーブル

主記憶

ページ枠	ページ
0	
1	
2	
3	
4	
5	
6	
7	

ページ化セグメンテーション

仮想アドレスの構成

<table>
<tr><td colspan="3">仮想アドレス</td></tr>
<tr><td rowspan="2">セグメント番号</td><td colspan="2">セグメント内相対アドレス</td></tr>
<tr><td>ページ番号</td><td>ページ内相対アドレス</td></tr>
</table>

アドレスが二重構造

例

01	1	321

セグメント内相対アドレス＝ページングの仮想アドレス

ページ化セグメンテーション

仮想アドレス

01	1	321

実アドレス

2	321

セグメントテーブル

番号	基底アドレス	セグメント長	フラグ V
00		900	0
01	6,000	1500	1
02	2,000	500	1

主記憶

ページ枠	ページ
0	
1	
2	
3	
4	
5	
6	
7	

セグメント01のページテーブル

ページ	ページ枠
0	7
1	2
2	5
3	0

ページ化セグメンテーションの利点

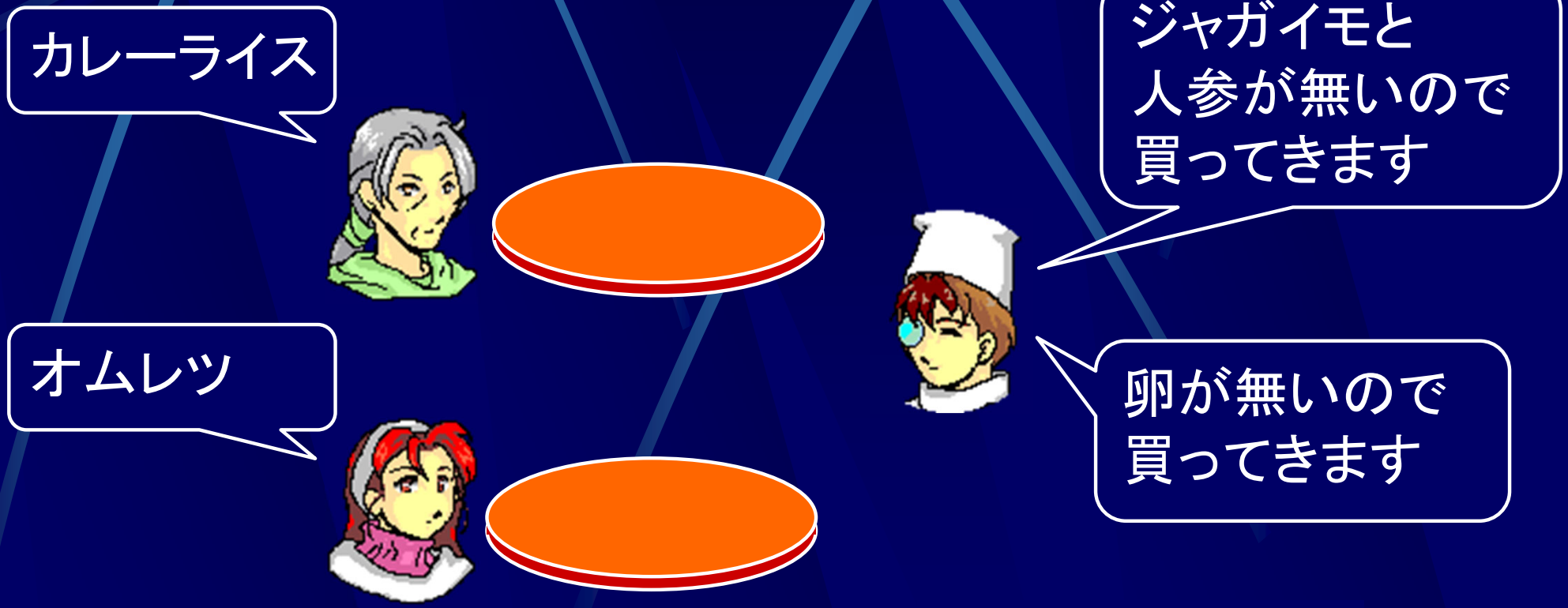
- 外部断片化の回避
  - ページ単位で主記憶割り当て
- 複数セグメント
  - 各セグメントは大きさ増減可能
  - 複数使用により用途別に使い分け可能
- プロセス間共有
  - セグメンテーションとほぼ同様に共有可能
- ページテーブルの分散
  - ページテーブルが複数に分割されるので，一部を仮想記憶に追い出すことで主記憶使用量削減

フェッチ技法(fetch)

- フェッチ技法
  - 要求時フェッチ(demand fetch)
    - プログラムが参照したときにデータを読み込む
  - プリフェッチ(prefetch)
    - 参照前に予めデータを読んでおく

フェッチ技法	ページング	食材
要求時フェッチ	必要としたときにページイン	毎回食事前に購入
プリフェッチ	必要なページを予測して予めページイン	1週間分のメニューを予測して週末に購入

要求時フェッチ(demand fetch)



注文が来てから
食材を買いに行く

要求時フェッチ(demand fetch)

必要になった時点でページイン

- 要求時フェッチの長所
  - 不必要なページをページインせずにすむ
  - ページ予測の必要無し
- 要求時フェッチの短所
  - 新しいページを参照する度にページイン発生
  - 処理・転送の並行動作が行えない

プリフェッチ(prefetch)

カレーライス

ジャガイモ100個
人参50本
卵100個を
仕入れておこう

すぐにお持ちします

オムレツ

すぐにお持ちします

きつねうどん

うどんと油揚げが
無いので
買ってきます。

プリフェッチ(prefetch)

暑くなりそうだから
アイスをたくさん
仕入れておこう

予想外に
寒くなって
売れなかった…

プリフェッチ(prefetch)

必要なページを予測して予めページイン

- プリフェッチの長所
  - プロセス実行時間が短縮(予測が正しければ)
  - CPU内部の処理と転送を並行に行える

- プリフェッチの必要条件
  - 予測の確度が高い
  - 予測処理が低コスト

トレードオフ

参照の局所性
(locality of reference)

- 参照の局所性(locality of reference)
  - 主記憶へのアクセスは一部のアドレスに集中する可能性が高い
- 時間局所性
  - 最近参照されたページは近い将来に再度参照される可能性が高い
- 空間局所性
  - あるページが参照されると近くのページも近い将来に参照される可能性が高い

参照の局所性

本Bの第1巻を
貸してください

本Bの第2巻を
貸してください

本Bの第3巻を
貸してください

本Aを
貸してください

本Aを
貸してください

時間局所性

- 時間局所性
  - 最近参照されたページは近い将来に再度参照される可能性が高い

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

for ループ内では
変数 *i*, *sum* が繰り返し参照される

空間局所性

- 空間局所性
  - あるページが参照されると近くのページも近い将来に参照される可能性が高い

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

for ループ内では
$a[0], a[1], \ldots, a[n]$ が順に参照される

局所性の原因

- プログラムの特徴
  - 関数, メソッドを用いて構造化
    - 関数内ではアクセスする記憶領域がほぼ同じ (時間&空間局所性)
  - プログラムの大部分はループで構成
    - ループ制御変数等、同一変数へのアクセス (時間局所性)
    - 配列等連続領域へのアクセス(空間局所性)

ページインしたページの近くのページも必要になる可能性が高い

プリフェッチ技法

- プリフェッチ技法
  - 要求時プリフェッチ(demand prefetch)
    - 多くのプログラムで1度参照したページの近傍のページが参照される

    ⇒ページを参照時に近傍のページも読み込む
  - 初期ロードプリフェッチ(initiate load prefetch)
    - プログラム開始直後はページフォルトが多発

    ⇒プログラム開始時に近傍のページも読み込む

まとめ

- 多重レベルページング
  - ページテーブルを多段化
  - 仮想アドレスのページ番号部を複数に分割
  - ページテーブル分割により主記憶使用量削減
    (必要の無いテーブルは2次記憶上に置く)

まとめ

- セグメンテーション
  - 仮想記憶と実記憶を対応させる可変長な単位
  - プロセスを意味のあるかたまりに分割
  - プロセス間のオーバラップ（共有領域）を許す
- ページ化セグメンテーション
  - ページングとセグメンテーションの利点を融合
  - 外部断片化を回避
  - 主記憶使用量の削減

まとめ

- フェッチ技法
  - 必要そうなページを予測して予め読み込む
    - 参照されたページの近傍のページも読む
    - プログラム開始時にプログラム先頭部分のページも読む