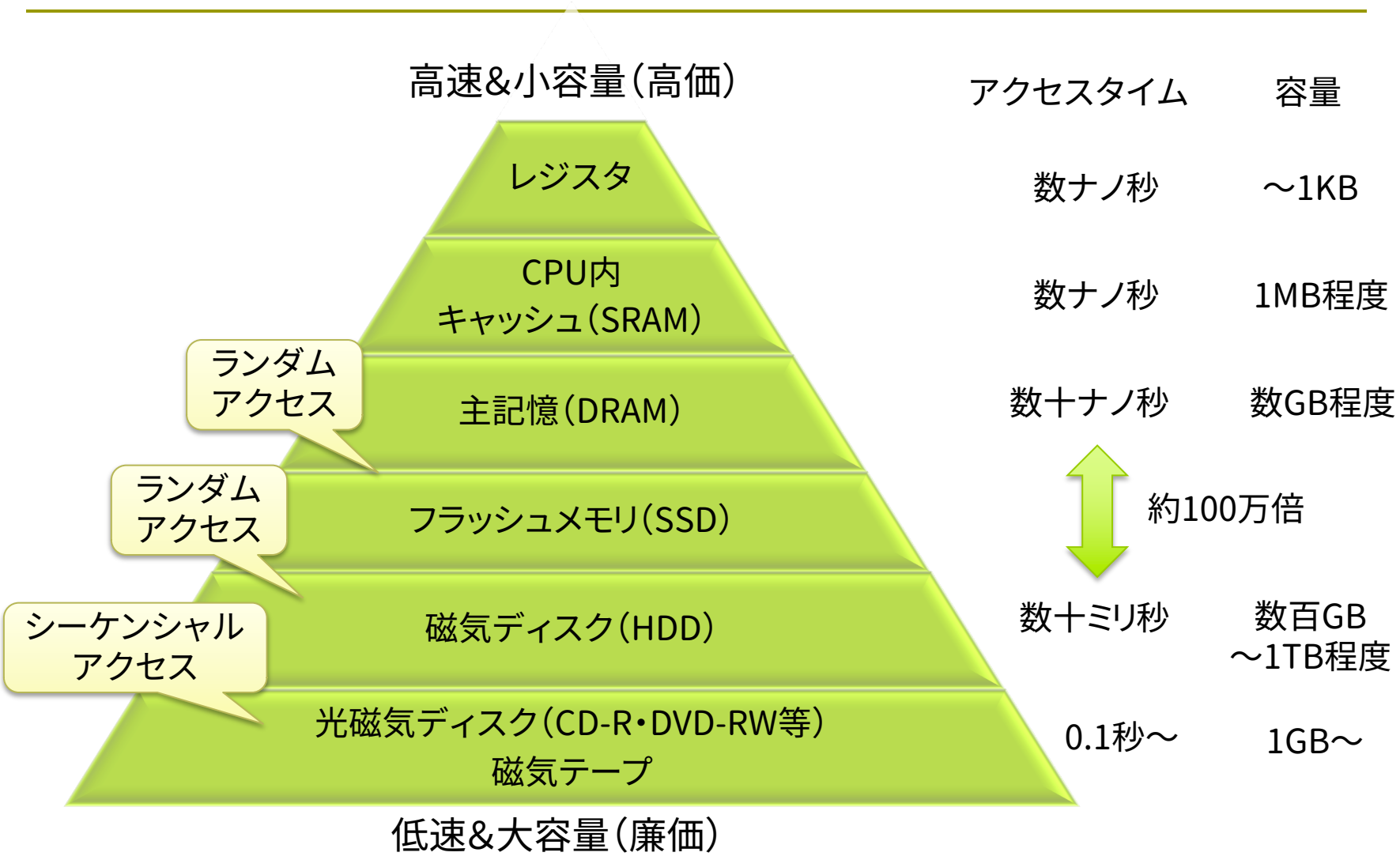
Operating Systems

仮想記憶

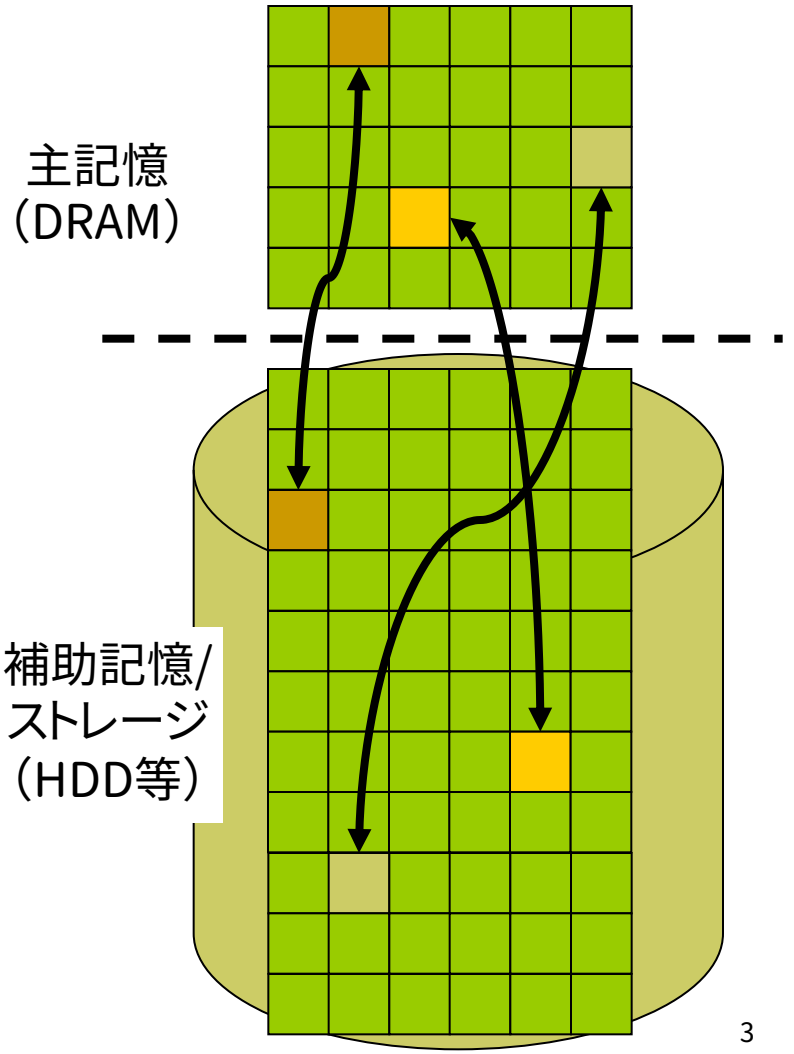
2020-11

記憶階層


高速&小容量(高価)
レジスタ
CPU内
キャッシュ(SRAM)
主記憶(DRAM)
フラッシュメモリ(SSD)
磁気ディスク(HDD)
光磁気ディスク(CD-R・DVD-RW等)
磁気テープ
低速&大容量(廉価)
ランダム
アクセス
ランダム
アクセス
シーケンシャル
アクセス
アクセスタイム
容量
数ナノ秒
～1KB
数ナノ秒
1MB程度
数十ナノ秒
数GB程度
約100万倍
数十ミリ秒
数百GB
～1TB程度
0.1秒～
1GB～

仮想記憶

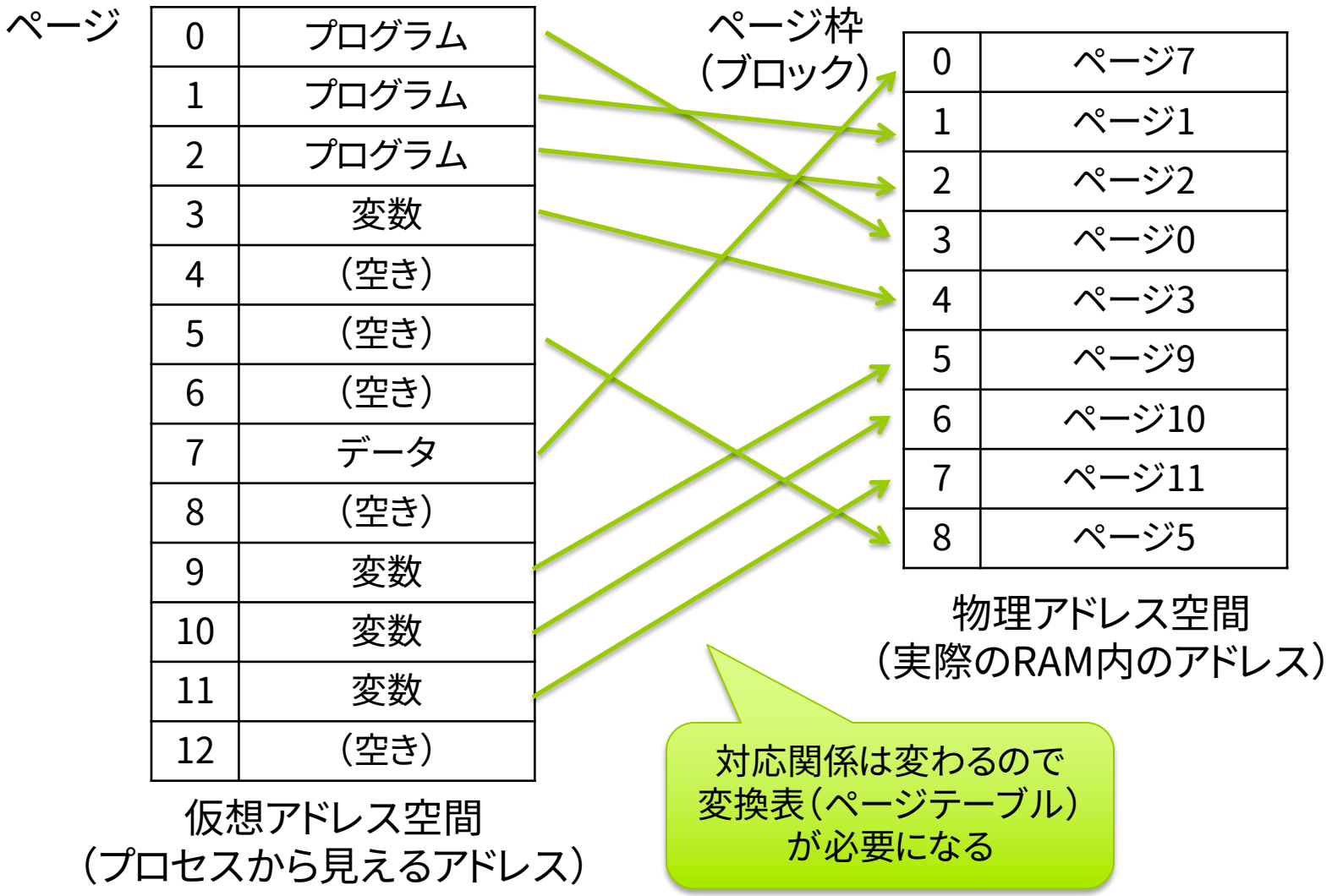
- 仮想記憶(virtual memory)
  - バーチャル=「実質的に同じ」
  - 補助記憶(HDD等)の領域を主記憶の延長のように見せかける技術
  - 使わないメモリ領域を補助記憶に待避し,使うときに主記憶に復帰する
- ページング方式
  - メモリ空間を固定長の区画(ページ)に自動的に分割して管理する
- セグメンテーション方式
  - プログラムごとに設定した可変長の区画(セグメント)で管理する



ページング

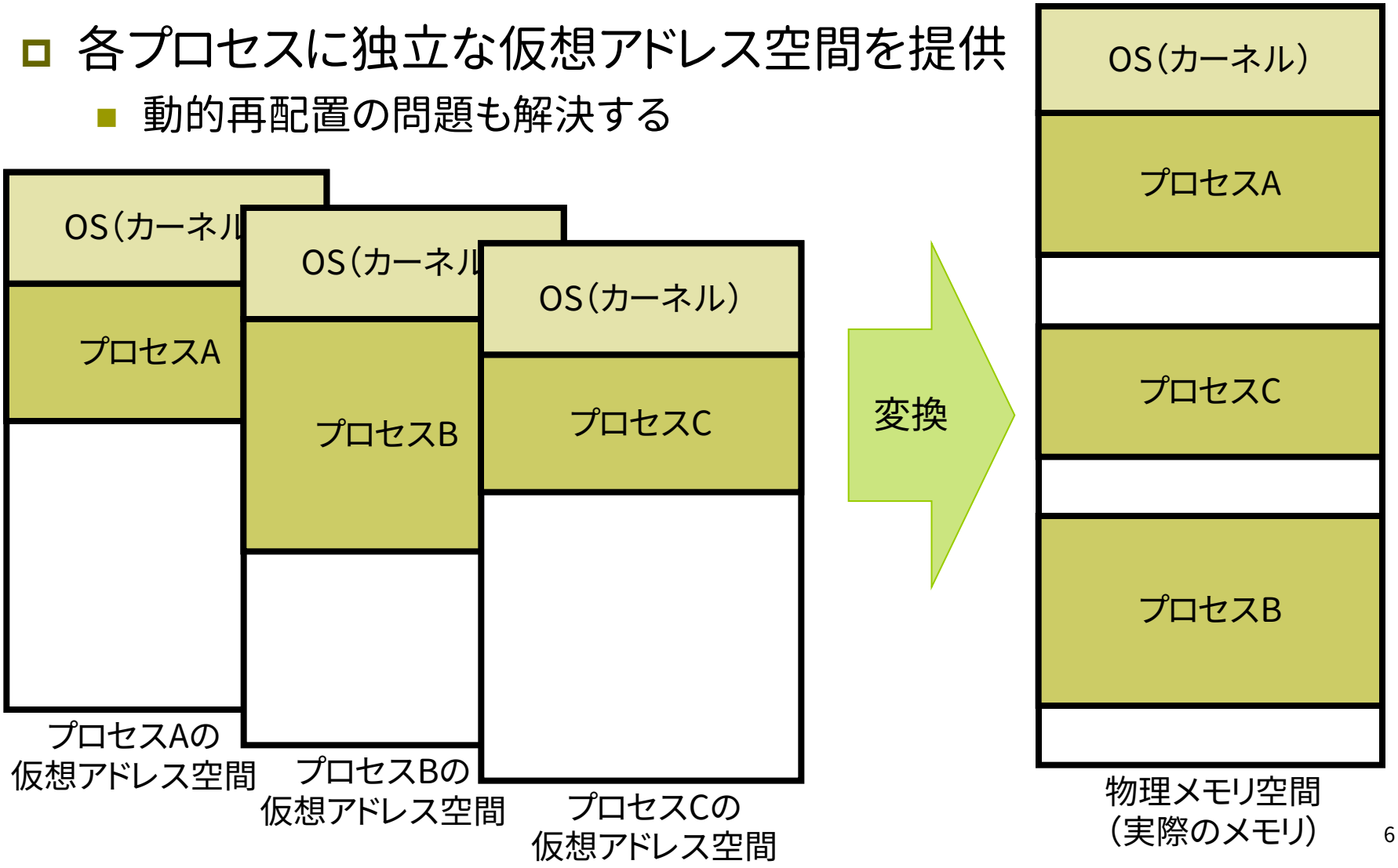
- メモリのページ化
  - 主記憶を，固定長（例：4Kバイト）のブロック（“ページ枠”）に分割する
  - メモリの内容（データ）も，同サイズの“ページ”の集合に分割する
  - どのページをどのページ枠に入れるか，自由に入れ替え可能にする
- アドレスの仮想化
  - 物理アドレス: ハードウェアメモリ内の位置 ← ページ“枠”のアドレス
  - 仮想アドレス: プロセスが実行中に使うアドレス ← ページのアドレス
  - プロセスが動くためには，「仮想」→「物理」の自動アドレス変換が必要
- ページングの効果
  - 物理メモリよりも広い仮想アドレス空間が使用できる ⇒ 仮想記憶
  - 物理的なメモリコンパクションやプロセスの再配置処理が不要になる

ページングの概念


ページ
0 プログラム
1 プログラム
2 プログラム
3 変数
4 (空き)
5 (空き)
6 (空き)
7 データ
8 (空き)
9 変数
10 変数
11 変数
12 (空き)
仮想アドレス空間
(プロセスから見えるアドレス)
ページ枠
(ブロック)
0 ページ7
1 ページ1
2 ページ2
3 ページ0
4 ページ3
5 ページ9
6 ページ10
7 ページ11
8 ページ5
物理アドレス空間
(実際のRAM内のアドレス)
対応関係は変わるので
変換表(ページテーブル)
が必要になる

多重仮想アドレス

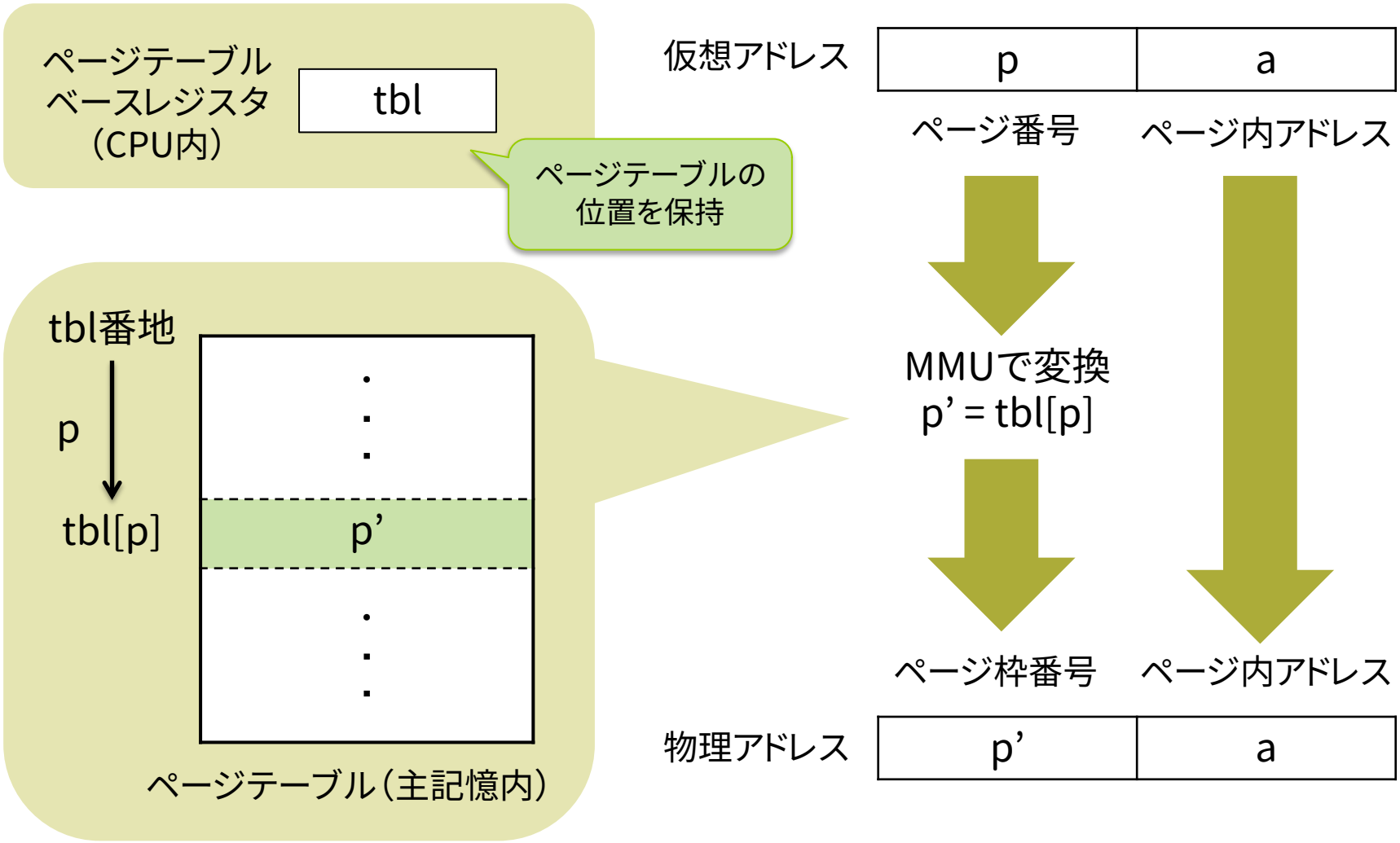
- 各プロセスに独立な仮想アドレス空間を提供
  - 動的再配置の問題も解決する



動的アドレス変換

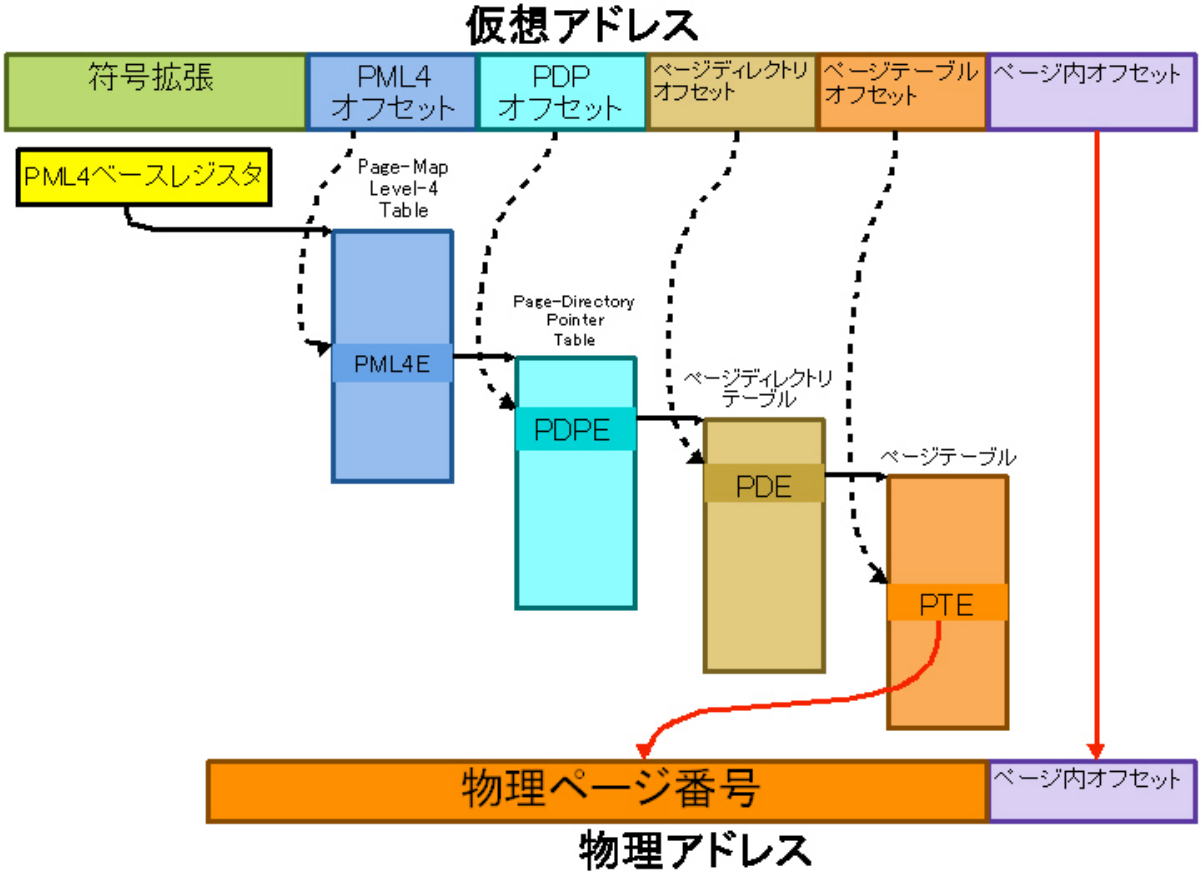
- 動的アドレス変換機構
  - プロセスが実行時に参照する仮想アドレスを，そのつど物理アドレス（ハードウェアメモリでのアドレス）に変換するしくみ
  - MMU（Memory Management Unit）というハードウェアが支援
- 各アドレスの内部形式
  - 32ビットなら，例えば上位20ビットと下位12ビット（4KB分）に分解
  - 例 0x12345678 → [ページ番号 0x12345] [ページ内 0x678]
  - 仮想アドレスの形式：[ページ　番号] [ページ内アドレス]
  - 物理アドレスの形式：[ページ枠番号] [ページ内アドレス]
- ページテーブル（アドレス変換テーブル）
  - 「ページ番号」→「ページ“枠”番号」という変換をするための表
  - 多重仮想アドレスの場合は，各プロセスがページテーブルを持つ

動的アドレス変換機構


ページテーブル
ベースレジスタ
（CPU内）
tbl
ページテーブルの
位置を保持
仮想アドレス
p
a
ページ番号
ページ内アドレス
MMUで変換
p’ = tbl[p]
tbl番地
p
tbl[p]
p’
ページテーブル（主記憶内）
ページ枠番号
ページ内アドレス
物理アドレス
p’
a

多段ページング

- 広大なアドレス空間に対応
  - 必要なときだけ2段目以降のテーブルを保持してメモリを節約



Wikipediaから引用

Intel X86のページング

表 3-3. ページサイズと物理アドレスサイズ

PG フラグ、CR0	PAE フラグ、CR4	PSE フラグ、CR4	PS フラグ、PDE	PSE-36 CPUID 機能フラグ	ページサイズ	物理アドレスサイズ
0	X	X	X	X	—	ページングはディスエーブル
1	0	0	X	X	4K バイト	32 ビット
1	0	1	0	X	4K バイト	32 ビット
1	0	1	1	0	4M バイト	32 ビット
1	0	1	1	1	4M バイト	36 ビット
1	1	X	0	X	4K バイト	36 ビット
1	1	X	1	X	2M バイト	36 ビット

『IA-32 インテル® アーキテクチャ ソフトウェア・デベロッパーズ・ マニュアル』から引用

Intel X86のページング（続き）

- 1段ページング（4Mバイト×2^{10}ページ）

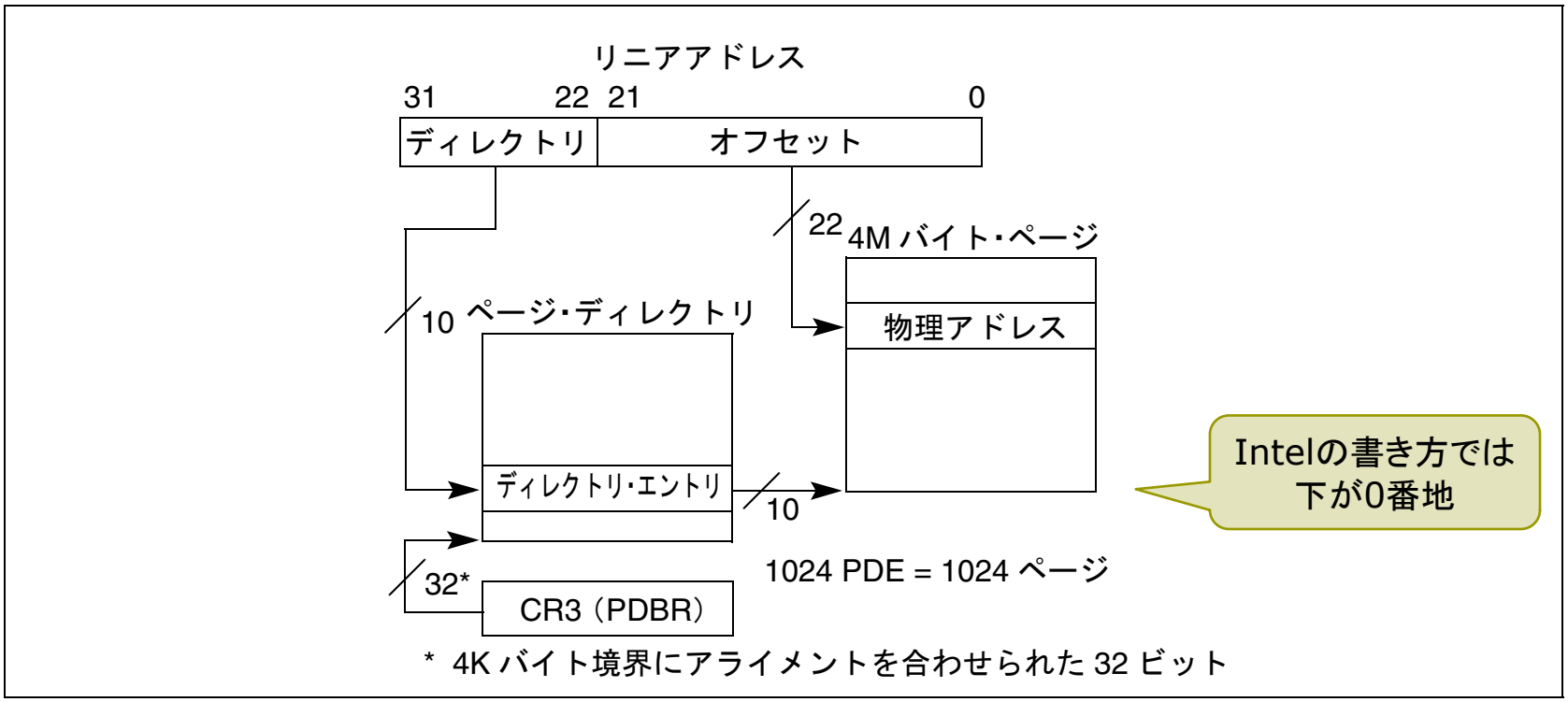


図 3-13. リニアアドレス変換（4M バイト・ページ）

『IA-32 インテル® アーキテクチャ ソフトウェア・デベロッパーズ・マニュアル』から引用

Intel X86のページング(続き)

- 2段ページング(4Kバイト×最大2^{20}ページ)

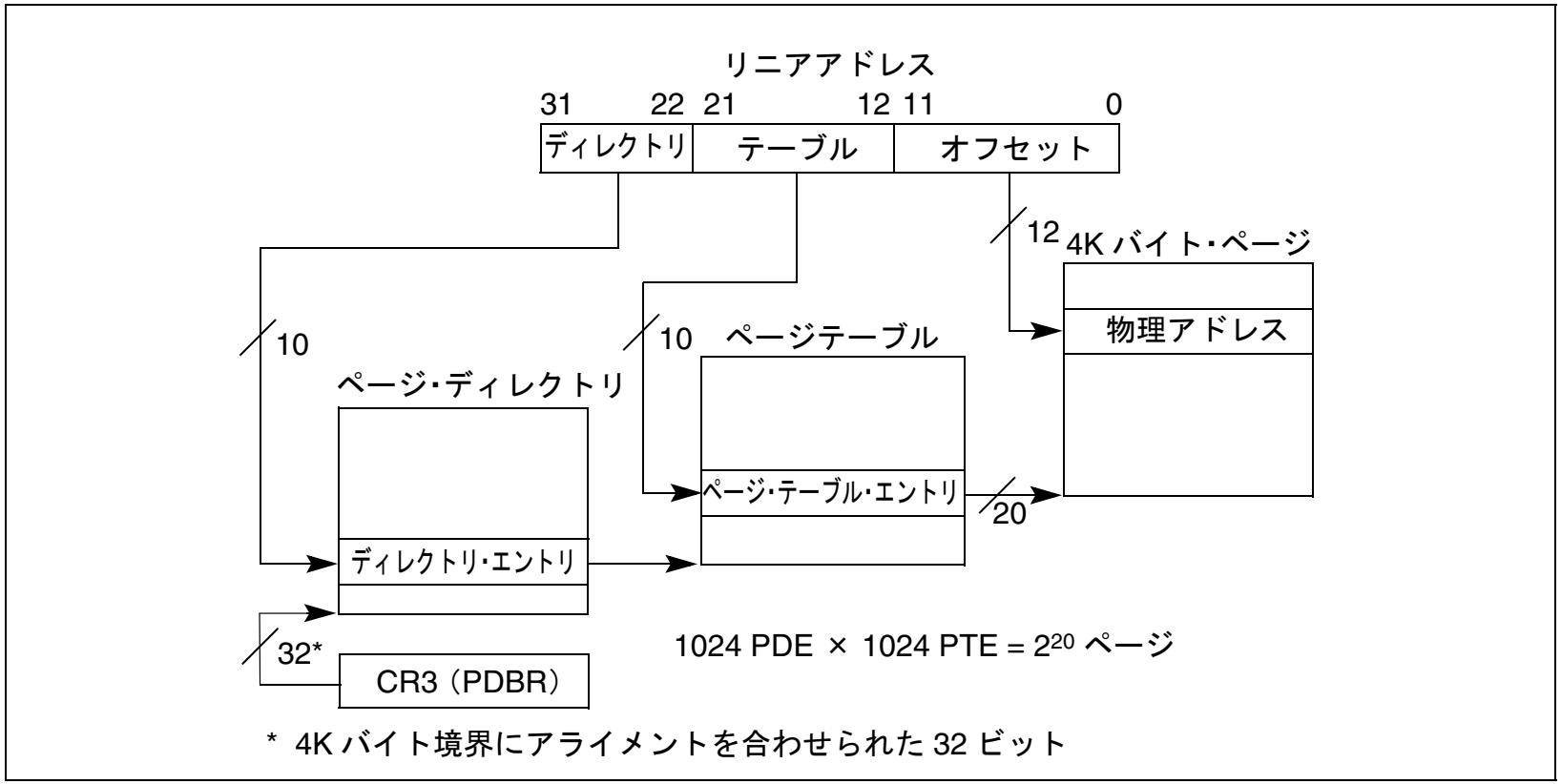


図 3-12. リニアアドレス変換（4K バイト・ページ）

『IA-32 インテル® アーキテクチャ ソフトウェア・デベロッパーズ・マニュアル』から引用

Intel X86のページング(続き)

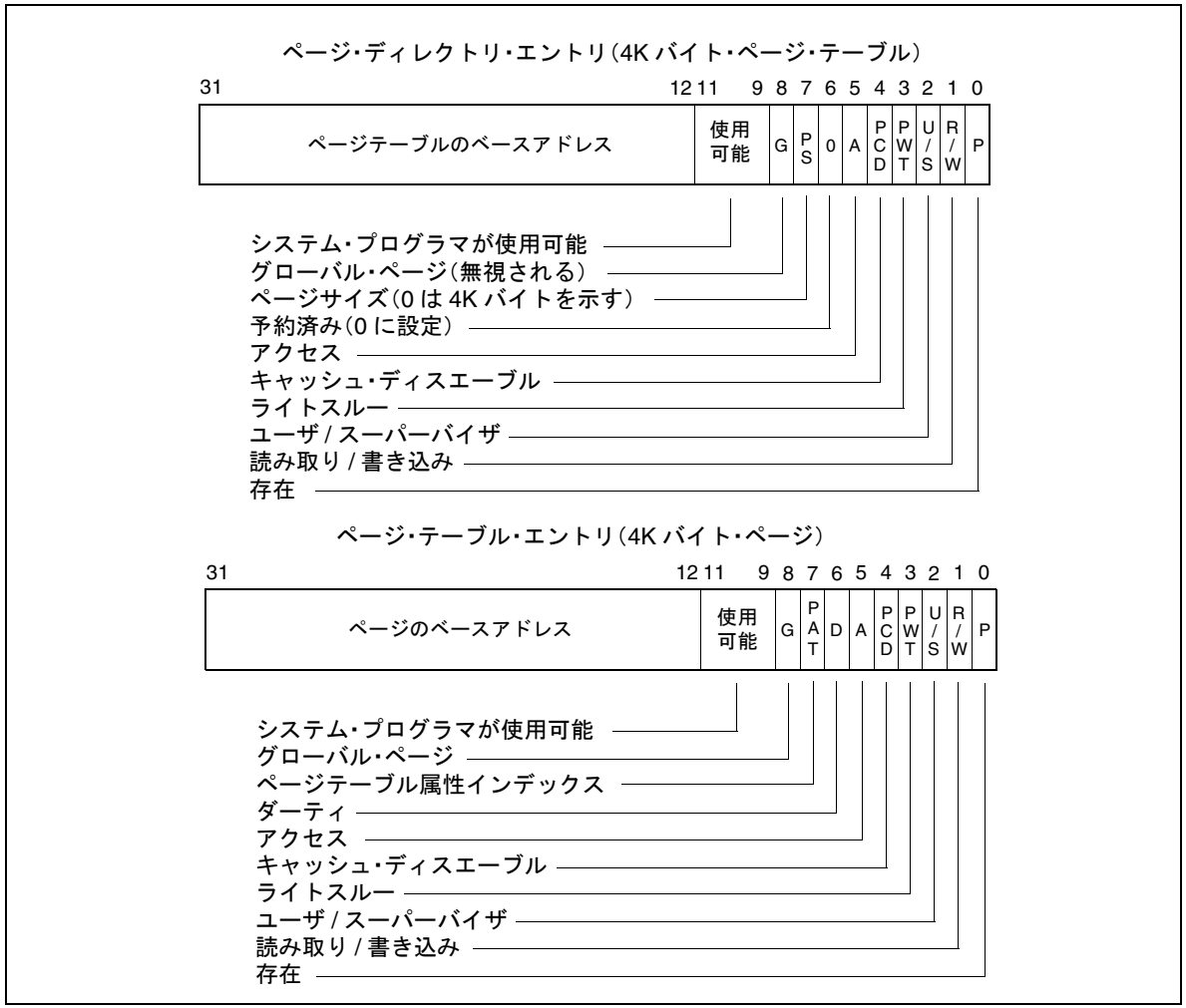


図 3-14. 4K バイト・ページと 32 ビット物理アドレスを使用する場合の
ページ・ディレクトリ・エントリとページ・テーブル・エントリのフォーマット

『IA-32 インテル® アーキテクチャ ソフトウェア・デベロッパーズ・マニュアル』から引用

Intel 64 / AMD64のページング

- 4段ページング(4Kページ×最大2^{36}ページ)

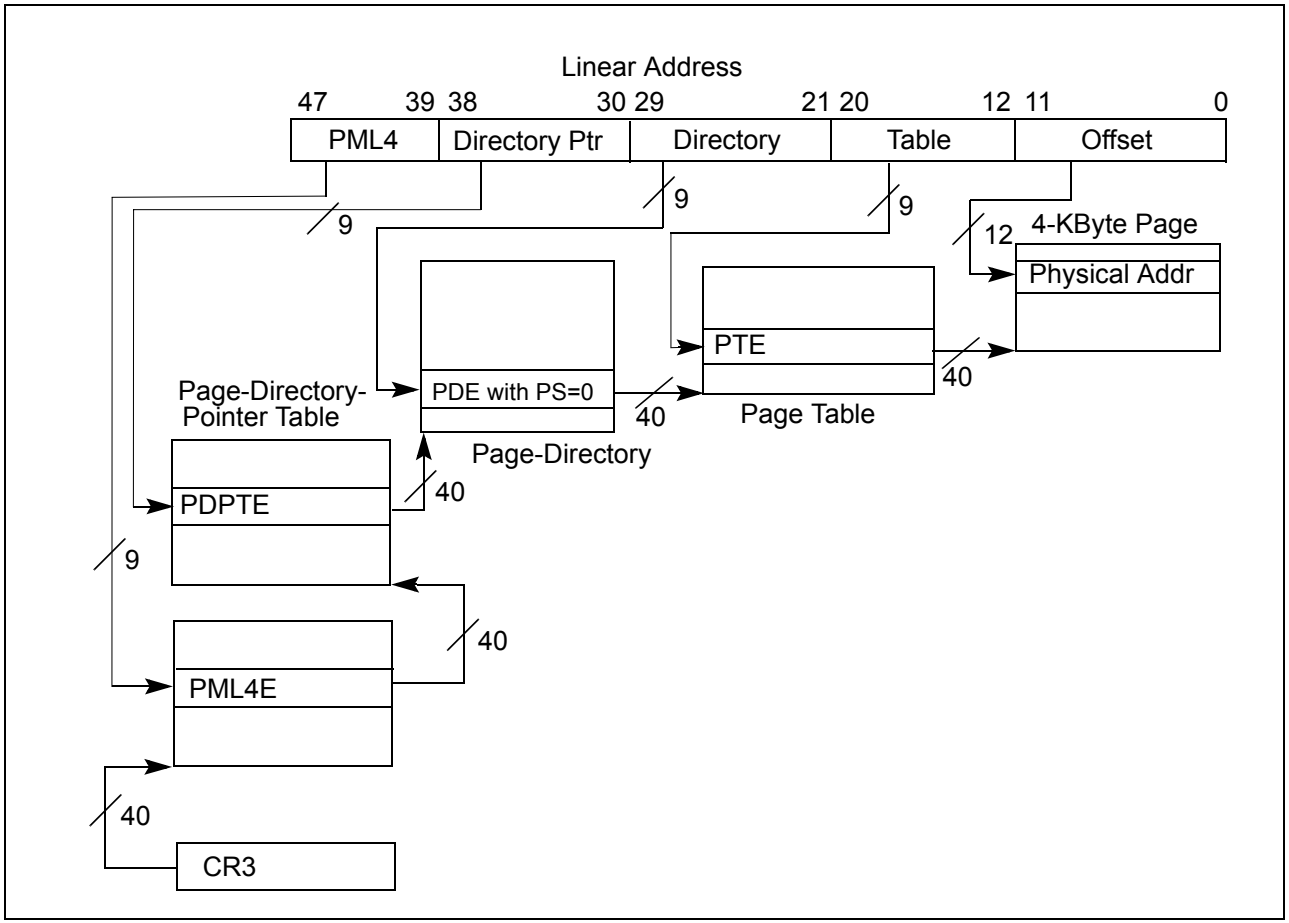
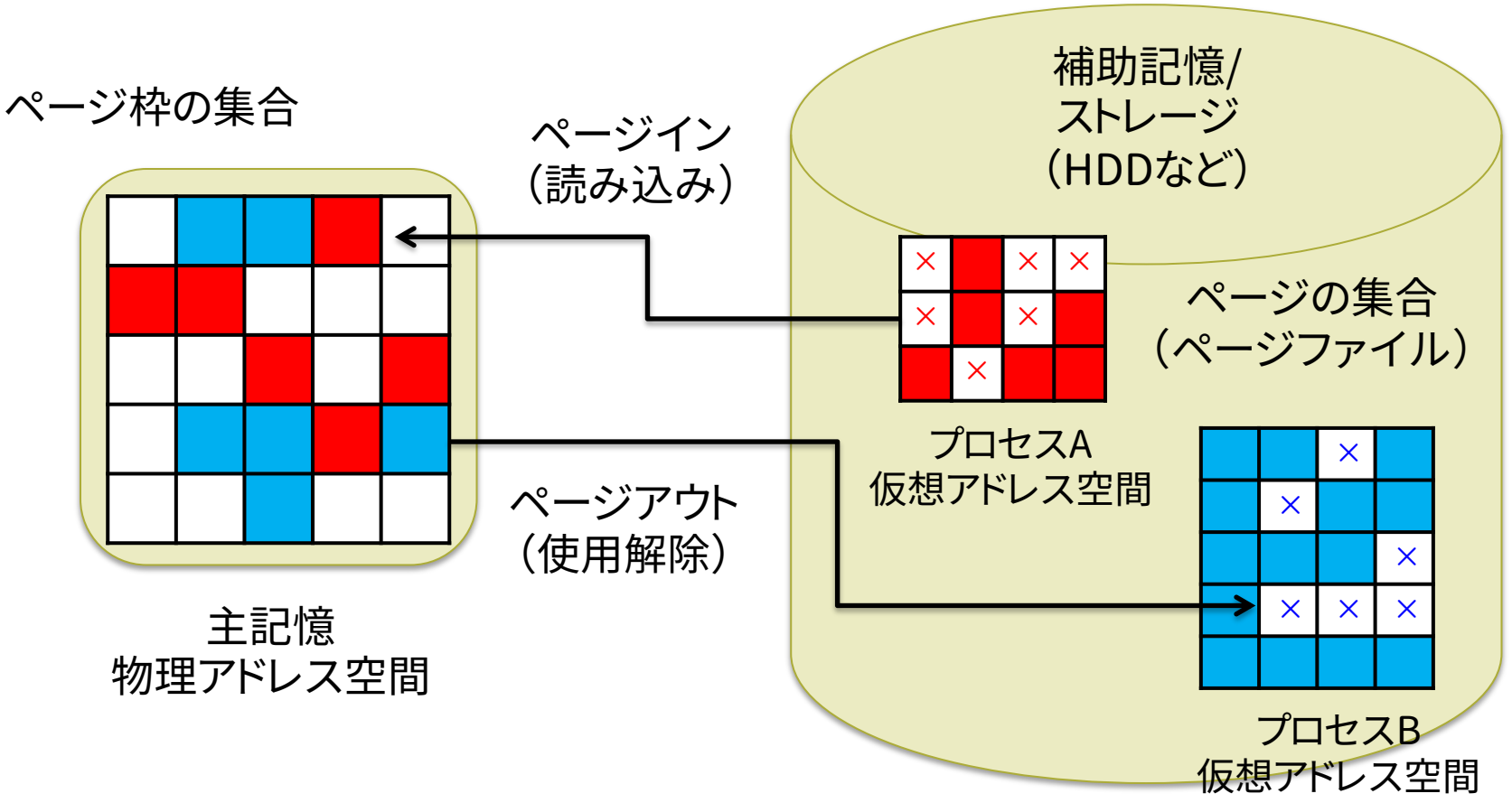


Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

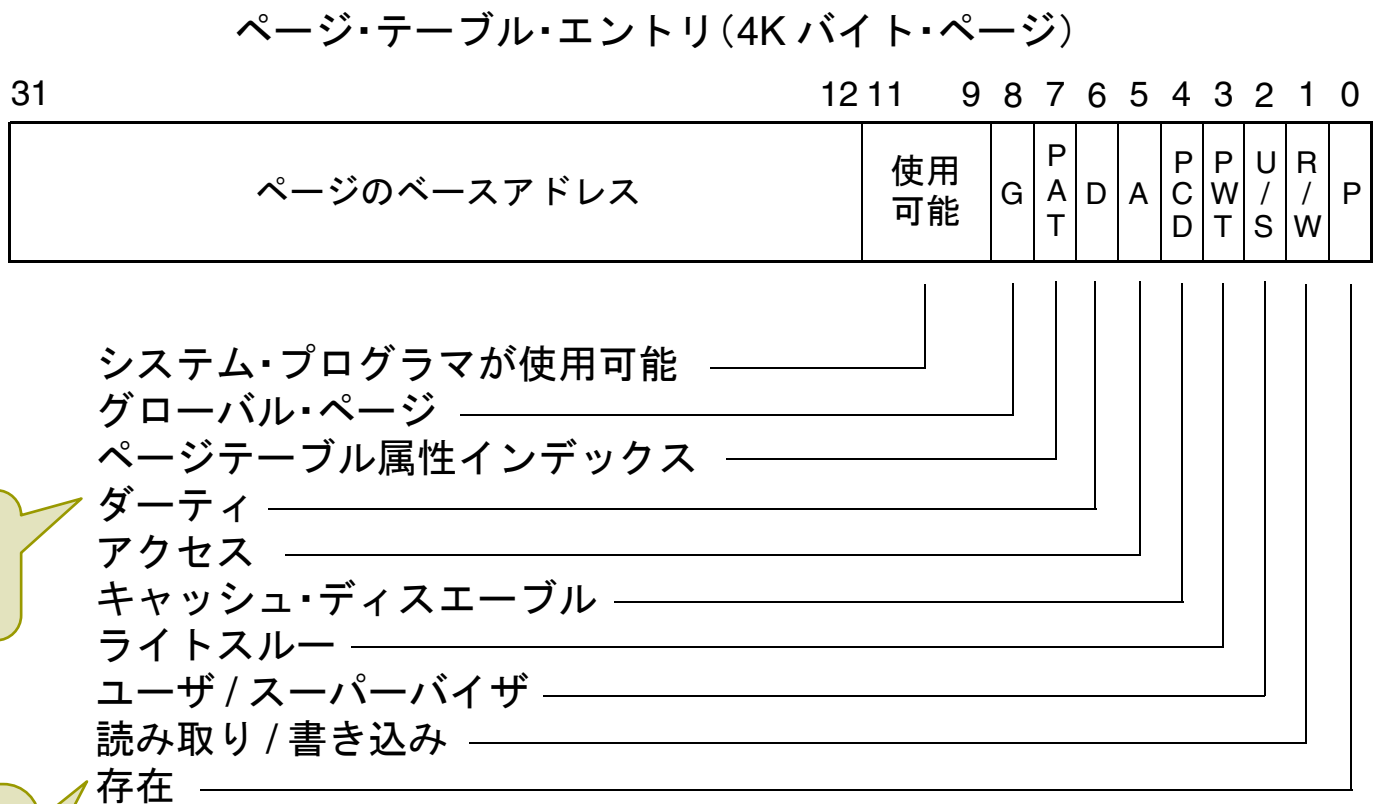
Intel® 64 and IA-32 Architectures Software Developer's Manual から引用

ページングによる仮想記憶

- 仮想アドレス空間の一部だけ，物理アドレス空間に保持する



ページテーブルのフラグ



Dirty:
内容変更済み

Present:
ページがメモリ内に存在

『IA-32 インテル® アーキテクチャ ソフトウェア・デベロッパーズ・ マニュアル』から引用

ページングによる仮想記憶の実現

- ページファイル
  - 主記憶に読み込みきれないページを保持しておくためのファイル（または独立のディスクパーティション）
  - Windowsの場合 C:¥pagefile.sys（デフォルトでは不可視設定）
  - 仮想記憶の容量 ≒ 物理メモリ容量 ＋ ページファイル容量
- ページフォールト
  - プロセスが仮想アドレスにアクセスしたとき，そのページに対応するページ枠がページテーブルにない（物理アドレス空間にない）こと
  - “例外トラップ”（一種の割り込み）というしくみで，OSに通知される
- ページの置き換え
  - カーネルのページ置き換え機能が，主記憶のページをどれかひとつ追い出して，仮想記憶のアクセスされたページと交換する

ページ置換方式

- ページ置換処理
  - ページアウト: ページ枠を空けるために，主記憶に読み込まれているどれかのページを補助記憶に退避すること
  - ページイン: 実行中のプロセスがアクセスしたページを補助記憶から主記憶のページ枠に読み込むこと

- ページ置換アルゴリズム
  - では，ページフォールトが起きたとき，どのページをアウトすべきか？
  - ランダム: 追い出すページは乱数で決める（性能比較の基準）
  - FIFO: 先に入ったページを，先に追い出す
  - OPT: 最適な方法，今後一番先まで使わないページを追い出す
  - LRU: 最も長い時間使われなかったページを追い出す
  - NRU: 一定時間使われなかったページからランダムに決める
  - LFU: 使用頻度（回数）が最低のページを追い出す

FIFOアルゴリズム

- 到着順ページ置き換え方式
  - First In, First Out（先入れ，先出し）
  - 先に入ったページを，先に追い出す
  - 性能はまあまあで，処理が簡単で高速である
  - Windowsはこれの改良版（らしい）

- FIFOアルゴリズムによるページングの例

ページ参照順		0	1	2	3	0	1	4	0	1	2	3	4
ページフォールト発生		v	v	v	v	v	v	v			v	v	
ページ枠の内容（主記憶）	0	0	0	0	3	3	3	4	4	4	4	4	4
	1		1	1	1	0	0	0	0	0	2	2	2
	2			2	2	2	1	1	1	1	1	3	3

OPTアルゴリズム

- 最適アルゴリズム
  - Optimum
  - “理論上”の最適な方法（他の方法の評価用）
  - 今後一番先まで使わないページを追い出す
  - 未来を予知していないと決められない ⇒ 実現不可能！

- OPTアルゴリズムによるページングの例

ページ参照順		0	1	2	3	0	1	4	0	1	2	3	4
ページフォールト発生		v	v	v	v			v			v	v	
ページ枠の内容（主記憶）	0	0	0	0	0	0	0	0	0	0	0	0	0
	1		1	1	1	1	1	1	1	1	2	3	3
	2			2	3	3	3	4	4	4	4	4	4

LRUアルゴリズム

- 最長不使用ページ置き換え方式
  - Least Recent Used
  - 最も長い時間使われなかったページを追い出す
  - 置換回数の性能は良いが，実現しようとすると処理が複雑で遅い
  - LinuxやMacは，これを簡単化したNRU（Not Recent Used）

- LRUアルゴリズムによるページングの例

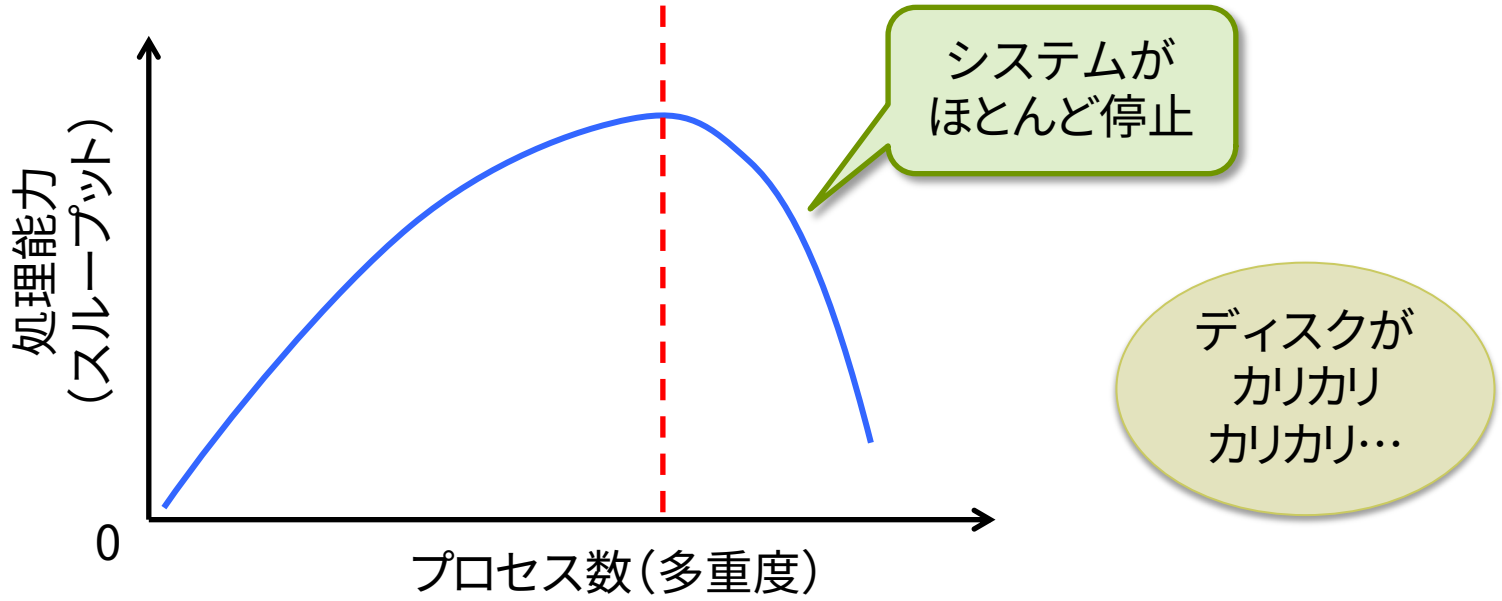
ページ参照順		0	1	2	3	0	1	4	0	1	2	3	4
ページフォールト発生		v	v	v	v	v	v	v			v	v	v
ページ枠の内容（主記憶）	0	0	0	0	3	3	3	4	4	4	2	2	2
	1		1	1	1	0	0	0	0	0	0	3	3
	2			2	2	2	1	1	1	1	1	1	4

参照の局所性

- 仮想記憶がうまく動く理由
  - 主記憶にプロセスの一部しか読み込まなくても，なぜうまく動くか？
  - プロセスが参照するアドレスは，空間的・時間的に偏る傾向がある
- 空間局所性
  - プログラムが同時に使うメモリは，（空間的に）近くにあることが多い
  - 例）逐次処理のプログラムコード，ローカル変数の配置など
  - ただし，オブジェクト指向プログラミングではそうならないことも多い
- 時間局所性
  - プログラムは，同じメモリを（時間的に）使い続けることが多い
  - ある時間を見ると，同じ領域のメモリを繰り返しアクセスしている
  - 例）ループ実行中のプログラムコード，関数内での変数アクセス

スラッシング

- スラッシング
  - OSがページ置換処理に忙殺され，他の処理が滞ってしまう状態
  - 多数のプロセスを実行させると，プロセス切り替えによって，頻繁にページフォールトが起こり，ページ置換処理が追いつかなるなる
  - 主記憶に対して仮想記憶を大きく取りすぎた場合に発生しやすい



仮想記憶の弱点

- 処理の遅延
  - ページ置換処理(HDD等からの読み込み)は時間がかかる
  - 特に,リアルタイム処理では,予測できない遅延は問題になる
- 処理が複雑
  - 専用のハードウェア機能(MMU)が不可欠
  - 組み込み用マイコンの多くでは,MMUの機能がない
  - ページ置換アルゴリズムも含めて,プログラムが複雑化する
- 補助記憶装置の使用が前提

多くの組み込みOS・ゲーム機では仮想記憶をサポートしない

演習課題

- 課題 11a ページ置換方式
  - 以下の表は，ページ枠が3つしかないコンピュータにおける仮想記憶のページ参照順を示している。
  - このページ参照順の場合に，FIFO，LRU，OPTの各ページ置換方式によって，どのようにページ枠の内容が変化するか，それぞれの表を完成させよ。
  - ページフォールトが発生する場合には，チェック印（v）を記入すること。

ページ参照順		0	1	3	0	2	0	3	4	2	0	1	2
ページフォールト発生													
ページ枠の内容（主記憶）	0												
	1												
	2												