オペレーティングシステム

2022/10/25

海谷 治彦

目次

- 前置き
- OSの構成 i386を想定して
  1. OSのためのハード機能
  2. 割り込み，マルチプログラミング
  3. カーネルについて
  4. 割り込み処理
  5. OSのカーネル以外の部分

まえおき: カーネル(OS)の役割

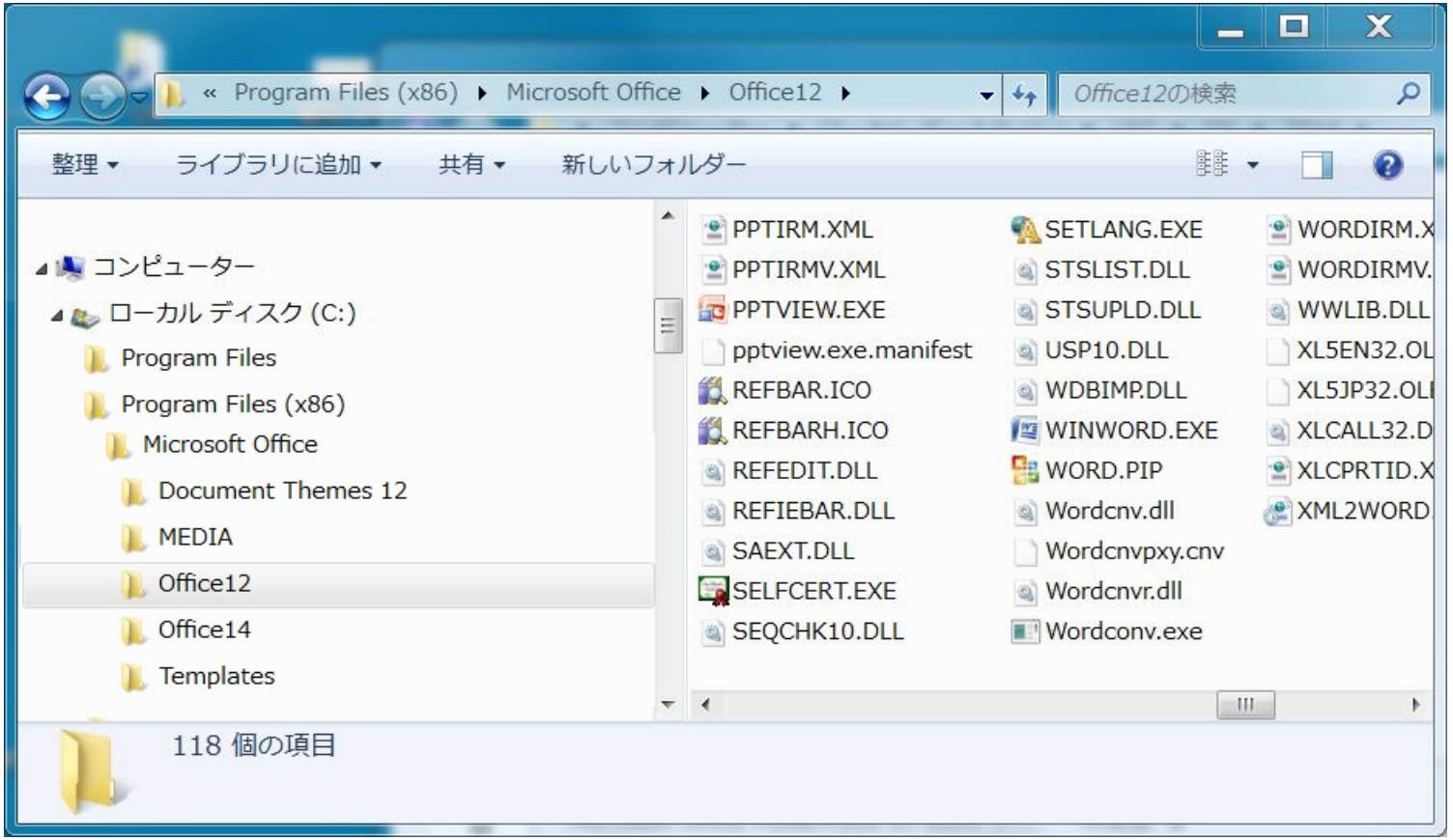
- カーネルはアプリの実行を支援する

- 個々のアプリ
  - アプリ(のプロセス)が必要な資源をカーネルが供給する.
  - カーネルがプロセス実行を開始し，終了する.
- アプリの多数のプロセス
  - カーネルがプロセス群の交通整理をする.
    - 必要な資源を排他的に共同利用
    - それぞれスムーズに動く

本日のストーリー

- 前頁「カーネルの役割」を，Linuxにおいて，とある具体的なCPUで実現する仕組みを解説する.
- とある具体的なCPU: Intel 80386
- 80386は本格的OSを実現可能とした最初のCPUの一つ．1985年誕生．i3, i5, i7 等の祖先.
- 32bit のバス幅を持つ
- 80386およびその類似CPUは，x86，IA-32などと呼ばれ，プログラムは現代でも現役で動いている.
- Intel社製.

今も現役 x86

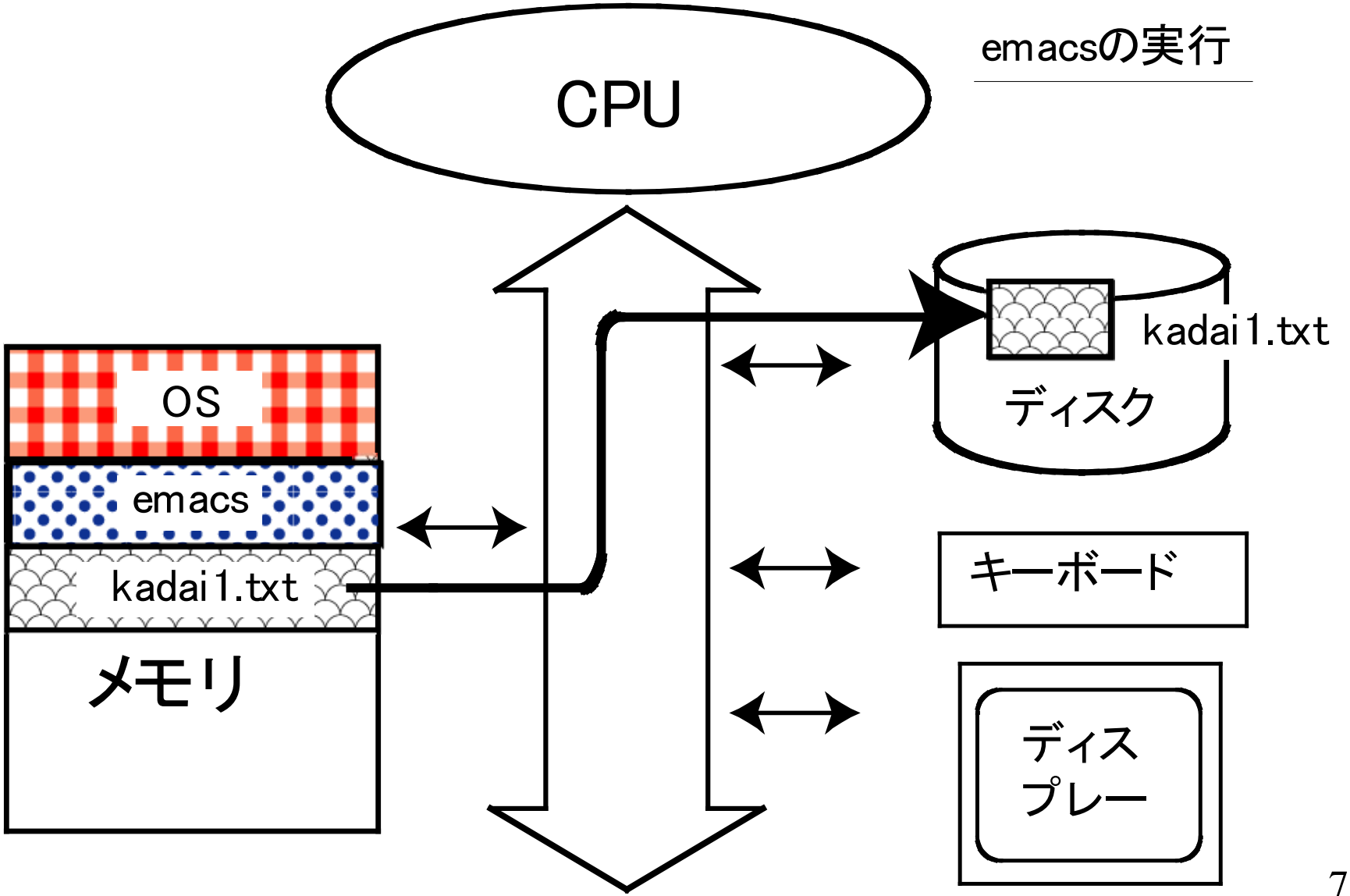


Program Files (x86) 下のプログラムは，80386互換プログラム
Win11でも同様.

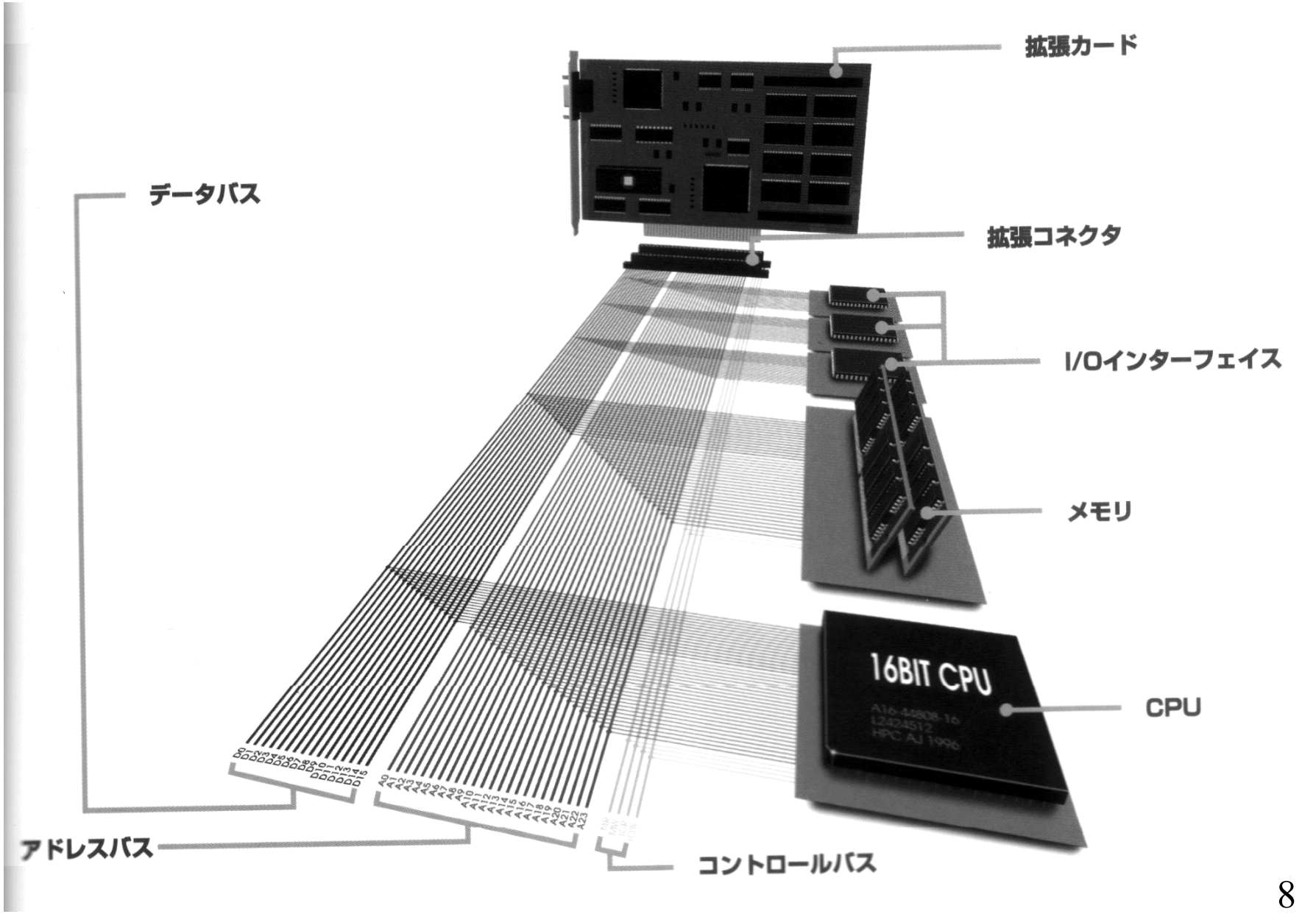
昨今のCPUの系譜

- x64
  - x86-64
  - Core i3, i5, i7 等，広く使われている 64bit CPU
  - x86と互換性があるインテルのCPU
- IA-64
  - インテルによる設計が一新された 64bitのCPU
  - x86と互換性が無い.
  - もう製造終了した.
- AMD Ryzen 等は x86-64 と互換
- ARM 等，違う系譜がスマホでは主流.

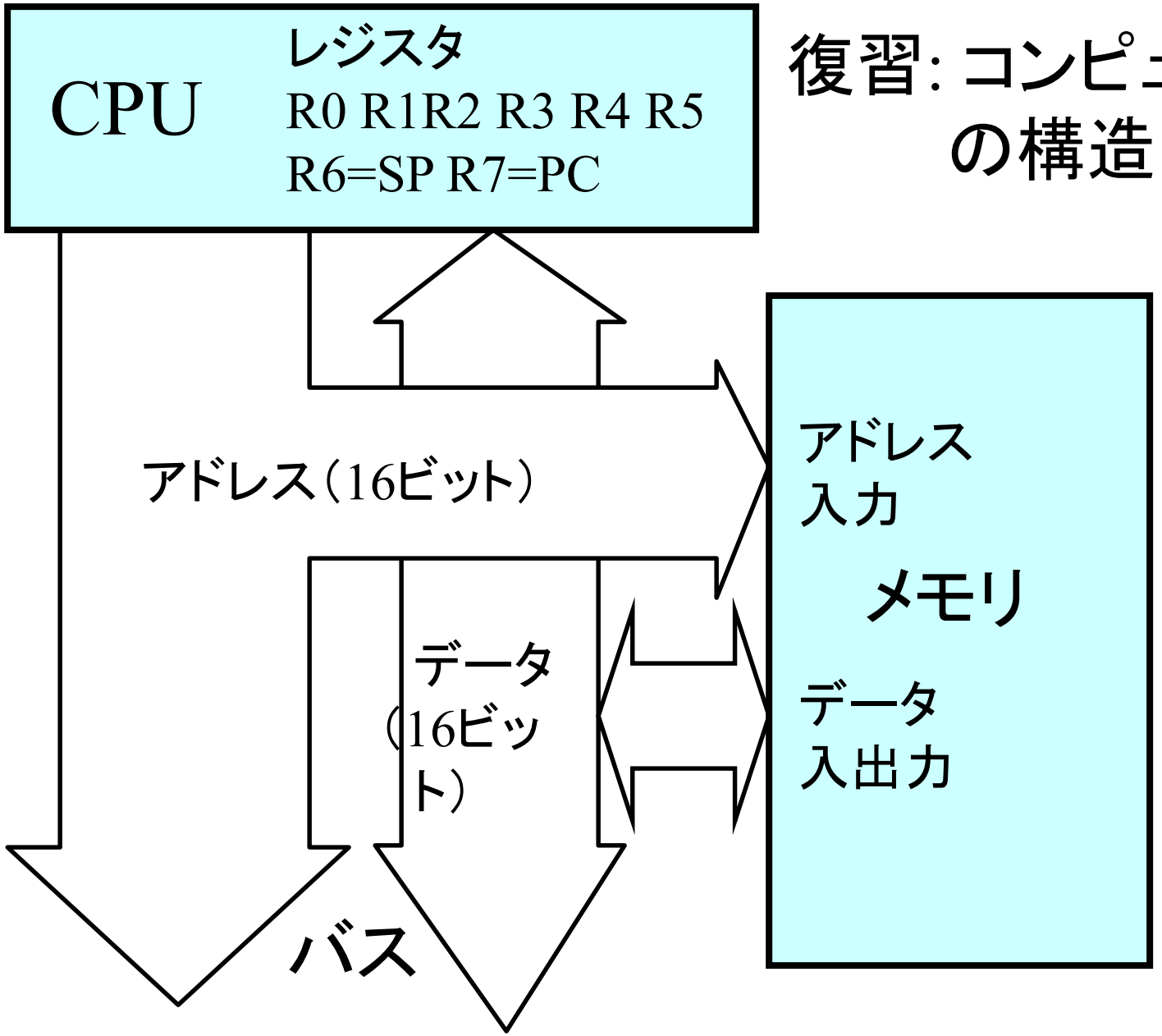
復習: ファイルとコマンド



復習: コンピュータの構造



復習: コンピュータの構造



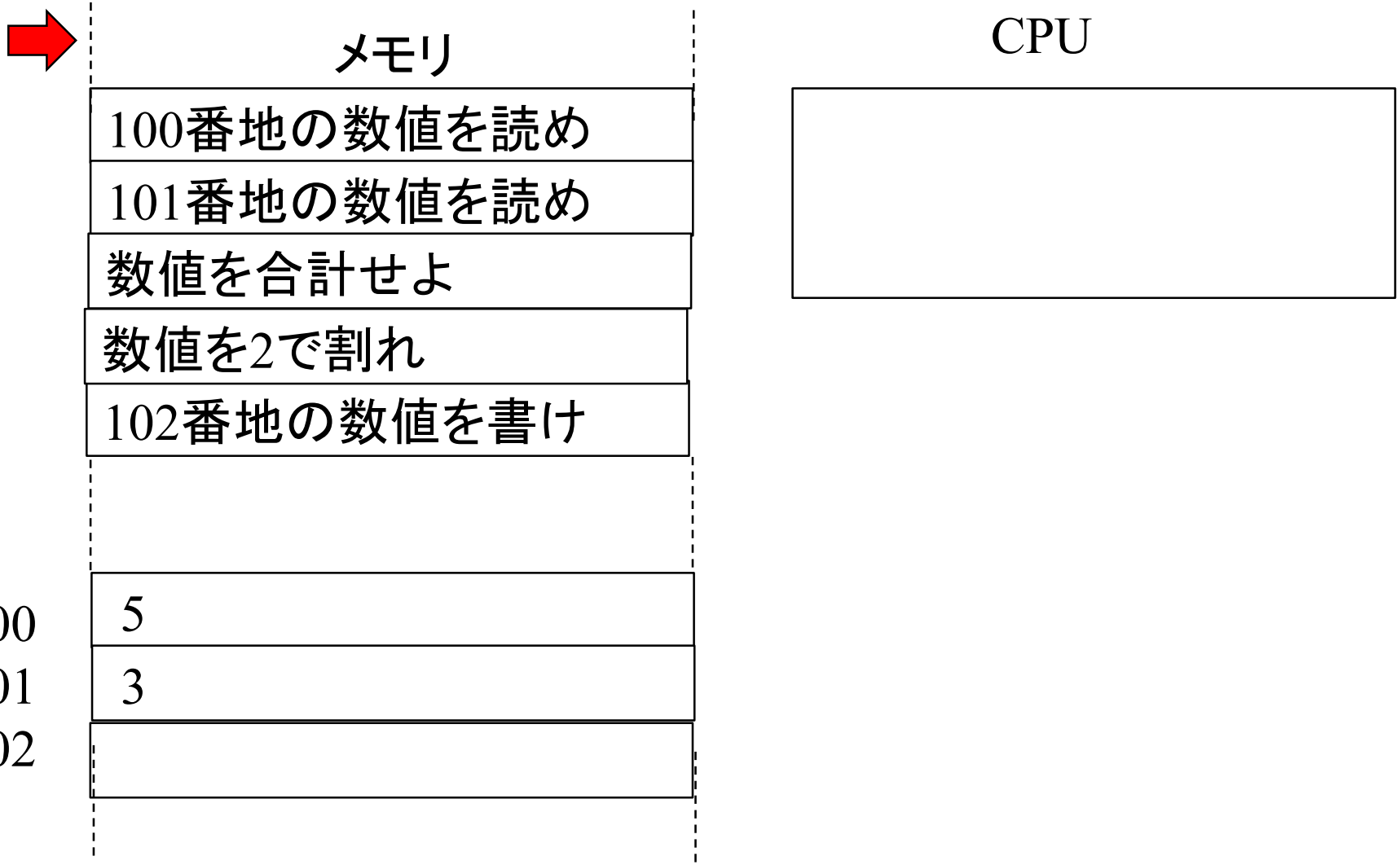
単一プロセスの基本動作

1. アドレスバスでメモリもしくはI/Oポートのアドレスを指定.
2. 指定した場所からデータをレジスタに読みこむ.
3. CPU内で, なんか計算する.
4. アドレスバスでメモリもしくはI/Oポートのアドレスを指定.
5. メモリもしくはI/Oポートに結果を書き出す.

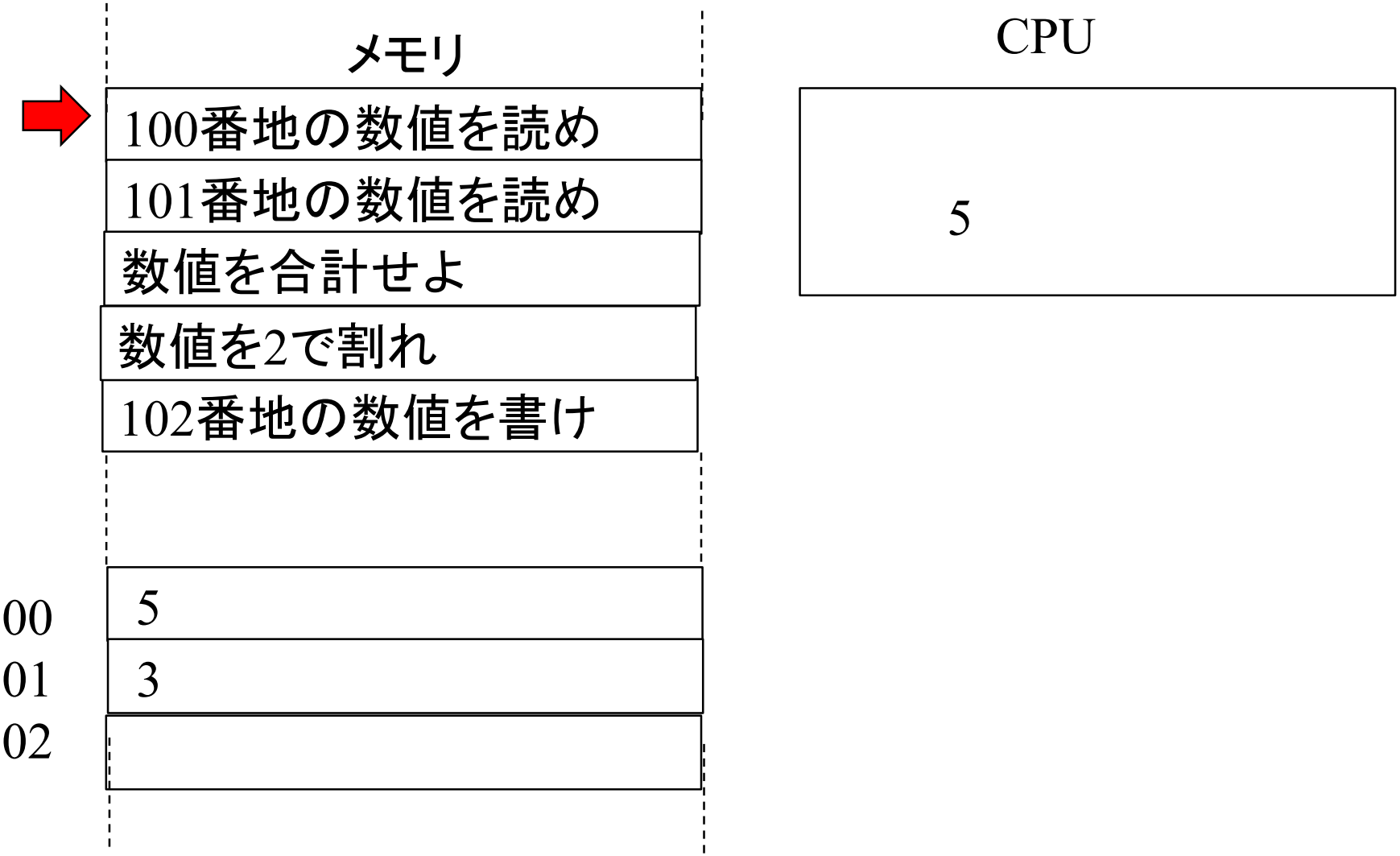
の繰り返し.

- プログラム自体もメモリに記録されており, 記述される順番に読んで実行するだけ.

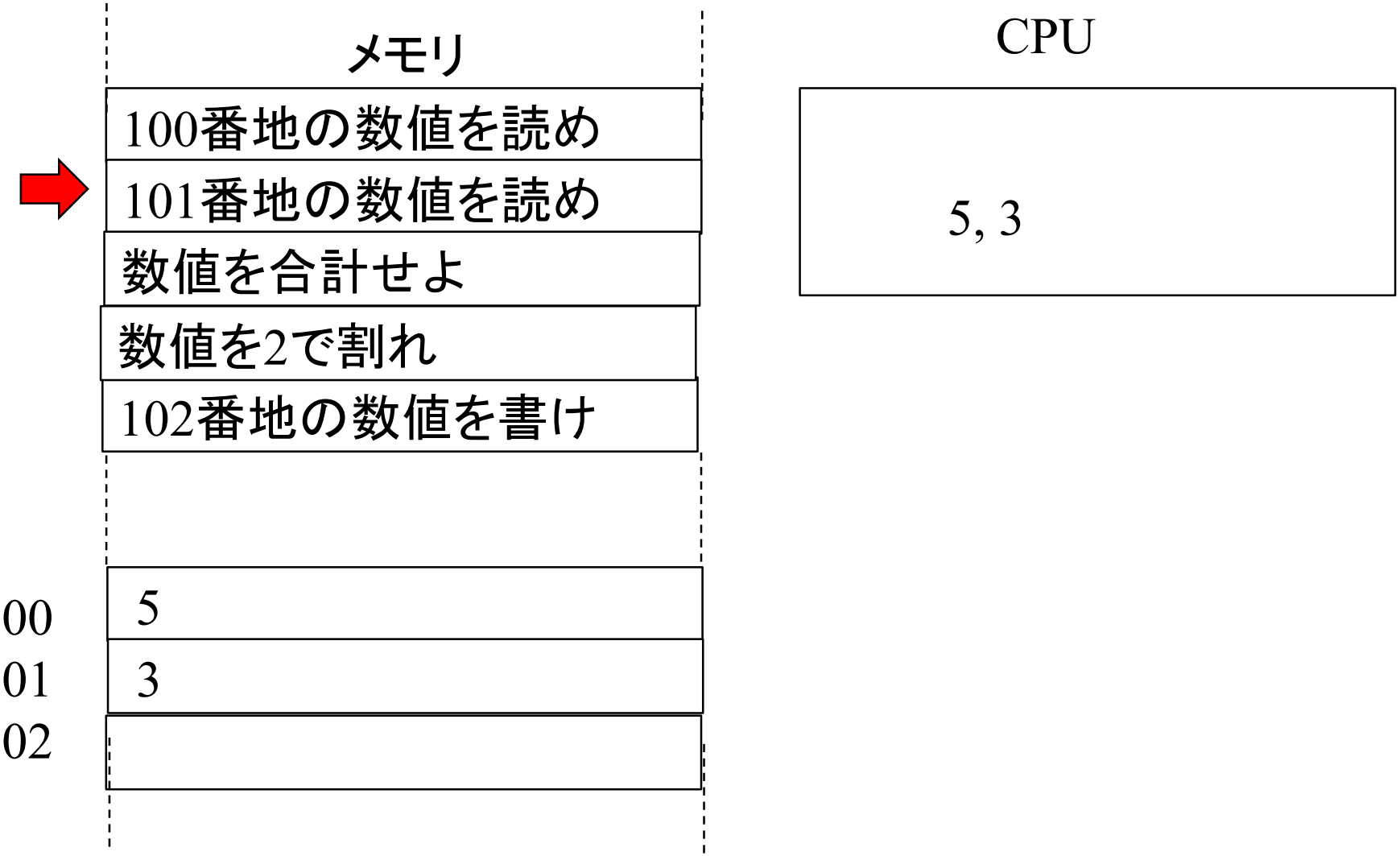
簡易な例題 ～ 二値の平均


メモリ
CPU
100番地の数値を読め
101番地の数値を読め
数値を合計せよ
数値を2で割れ
102番地の数値を書け
100
101
102
5
3

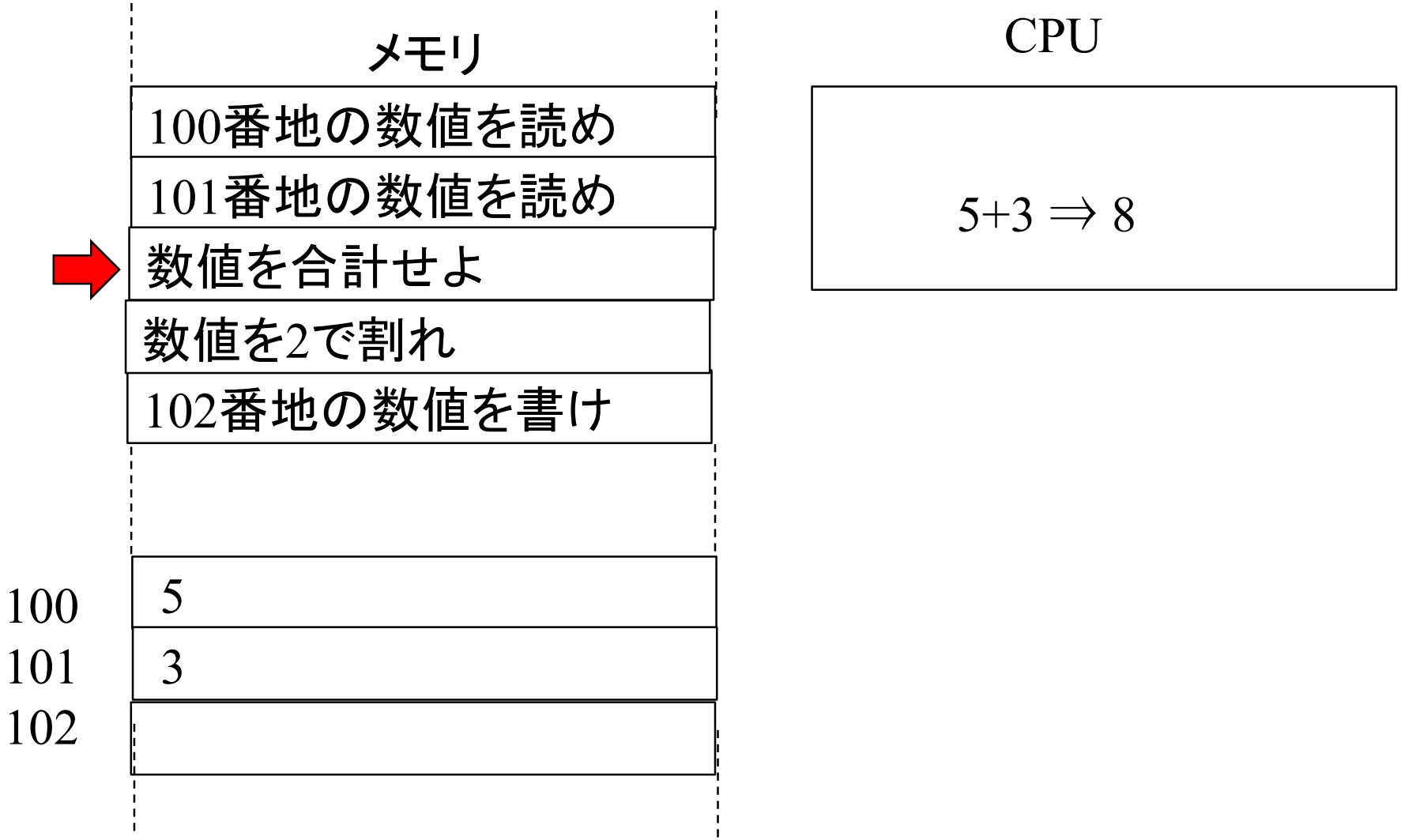
簡易な例題 ～ 二値の平均


メモリ
CPU
100番地の数値を読め
101番地の数値を読め
数値を合計せよ
数値を2で割れ
102番地の数値を書け
5
100
5
101
3
102

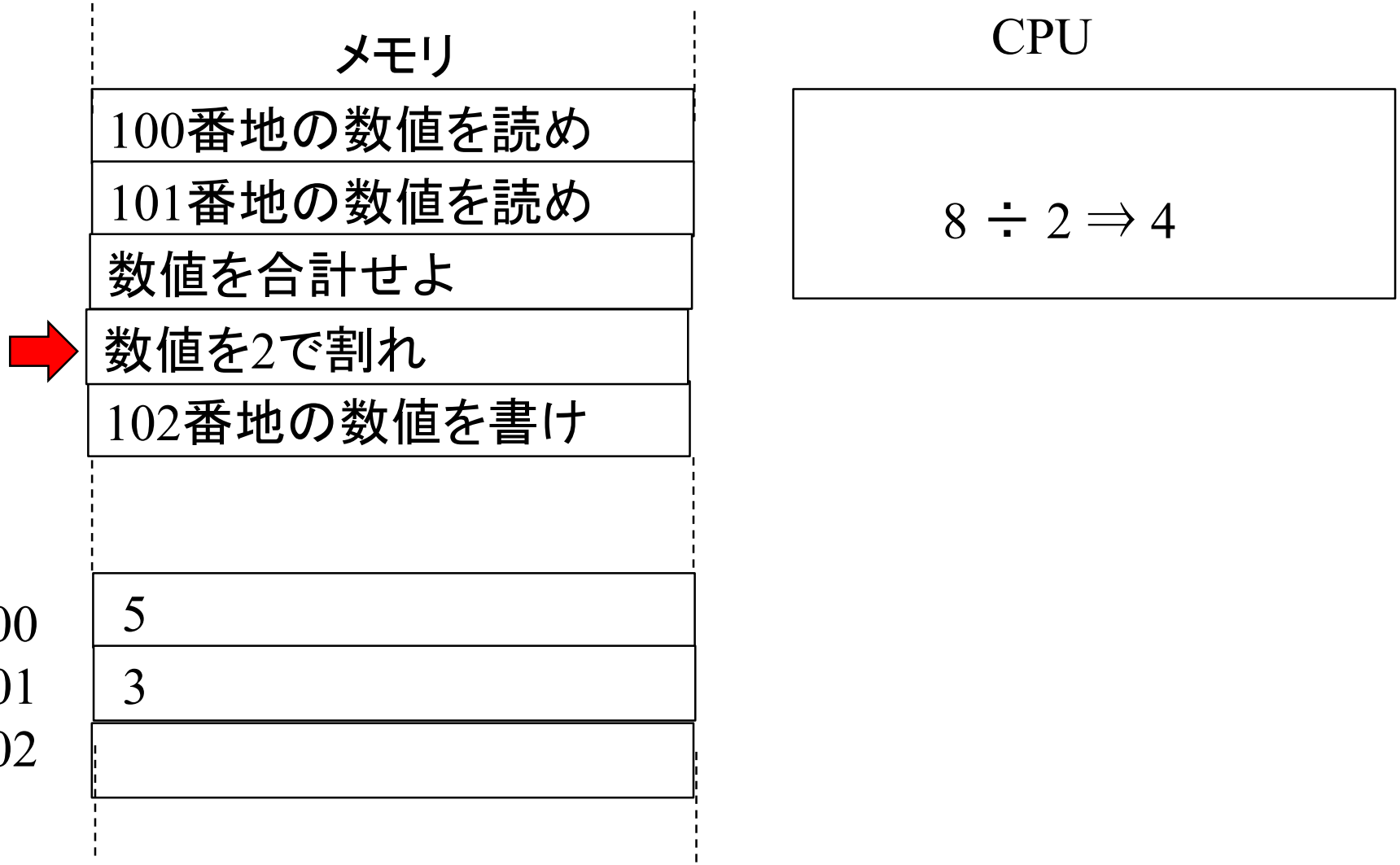
簡易な例題 ～ 二値の平均


メモリ
CPU
100番地の数値を読め
101番地の数値を読め
数値を合計せよ
数値を2で割れ
102番地の数値を書け
5, 3
100
101
102
5
3

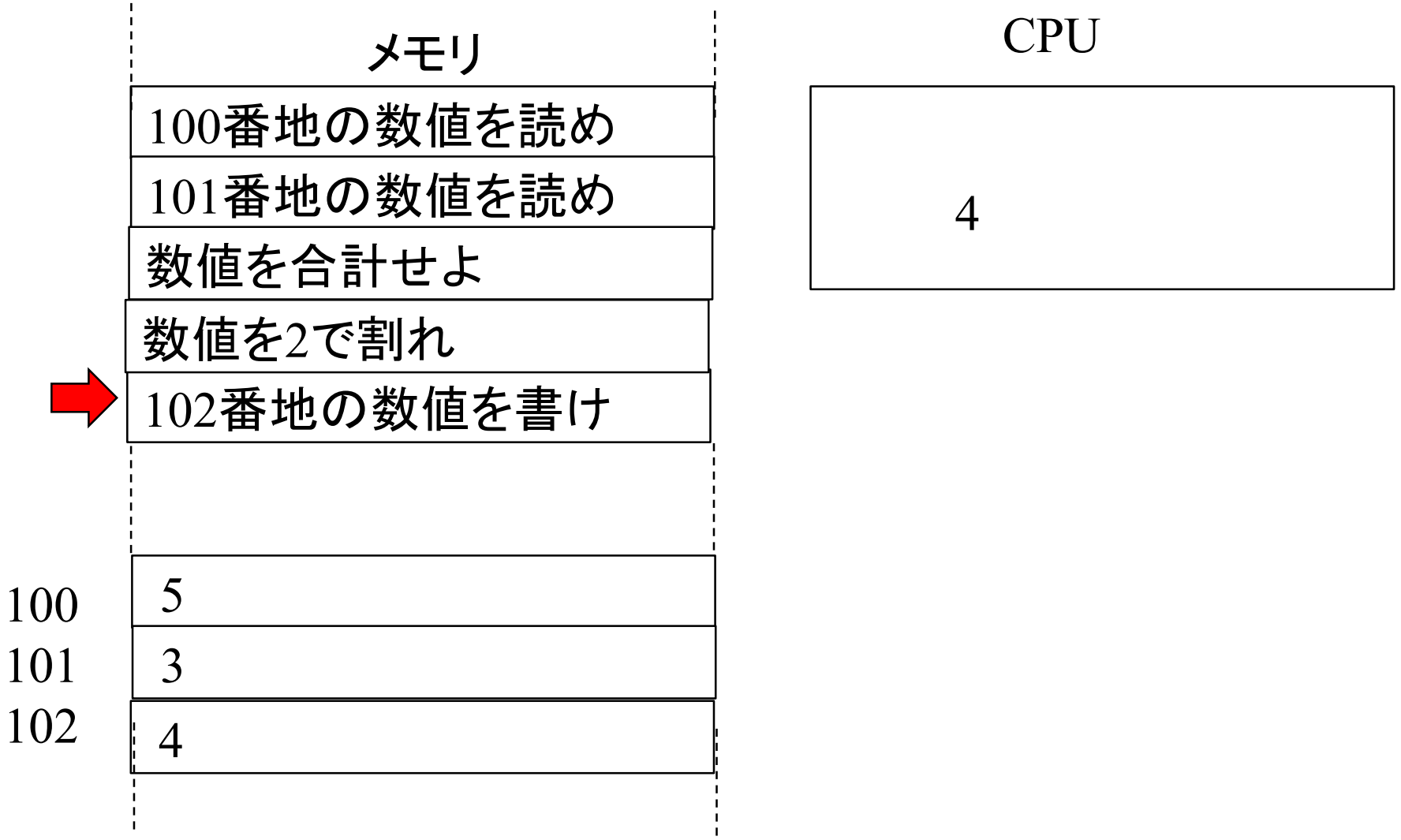
簡易な例題 ～ 二値の平均


メモリ
100番地の数値を読め
101番地の数値を読め
数値を合計せよ
数値を2で割れ
102番地の数値を書け
100
101
102
5
3
CPU
5+3 ⇒ 8

簡易な例題 ～ 二値の平均


メモリ
100番地の数値を読め
101番地の数値を読め
数値を合計せよ
数値を2で割れ
102番地の数値を書け
100
101
102
5
3
CPU
8 ÷ 2 ⇒ 4

簡易な例題 ～ 二値の平均


メモリ
CPU
100番地の数値を読め
101番地の数値を読め
数値を合計せよ
数値を2で割れ
102番地の数値を書け
4
100
101
102
5
3
4

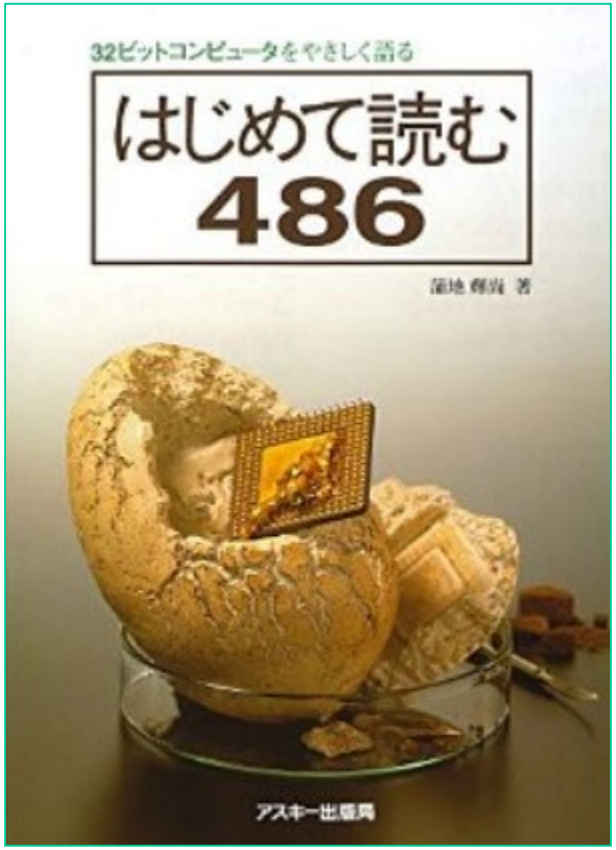
コードとデータ

- あるプロセス(OS自信の場合も)が動作する場合，
- プログラムがメモリに格納されている部分がある.
  - 以降，「コードセグメント (CS)」と呼びます.
- データを格納する部分がある.
  - 以降，「データセグメント (DS)」と呼びます.

- セグメント segment 区間，部分という意味

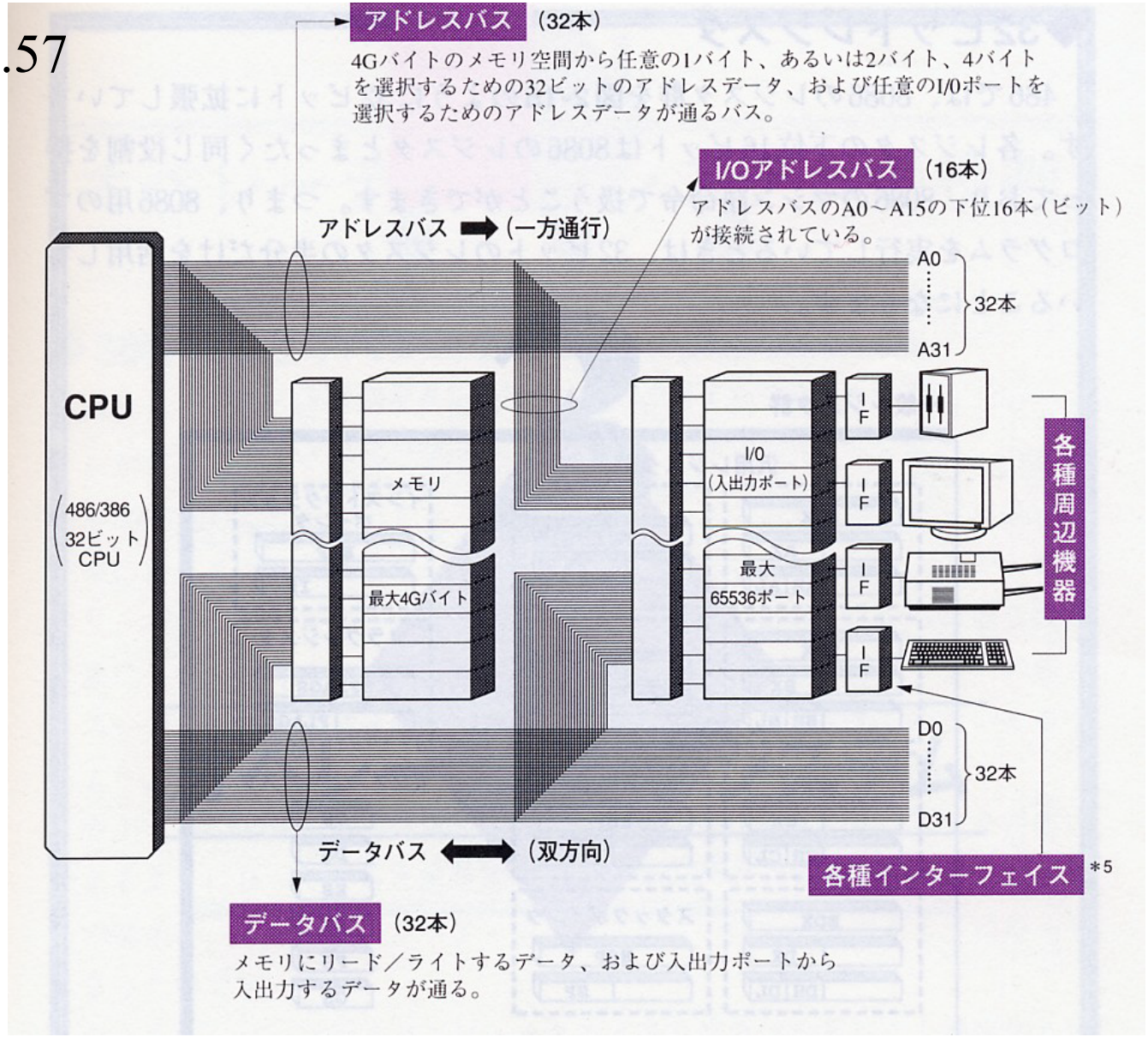
ネタ本

- 名著，ただし絶版.
- アマゾンでは中古で手に入る！
  - Kindle版もある模様



実際の386CPUの構造

文献 p.57



参考: アドレスバス本数とメモリの広さ

$2^{16} = 65536 \fallingdotseq 65K$

$2^{32} = 4294967296 \fallingdotseq 4G$

$2^{64} = 18446744073709551616$

$\fallingdotseq 18E$ (エクサ)

32bitマシン主流の時代,
4G以上のメモリが無駄
といわれた所以は上記にある.

参考: バス本数が3本の場合
(2^3=8) 8個のアドレスしか
指定(区別)できない.

1本	2本	3本
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

CPUの 32bit 64bit

- CPUについて上記の区別が使われる.
- 何を基準としているかたまに曖昧.
- 以下の三種類の基準があり意味合いが異なる.
- アドレスバスの本数 (アドレス表記の桁数)
  - 扱えるメモリ数(アドレス上限)に影響
- データバスの本数
  - データ交換の早さに影響
- レジスタの大きさ
  - 一度に計算できる数の大きさに影響
  - マシン語命令の違いもコレに由来

OS の 32bit 64bit

- ほぼ，intel x86 と x86_64 との対比で使われる.
- 32bit OS は 32bit時代からある古いマシン語の命令のみで構成される.
- 64bit OS は 64bit化された後の新しいマシン語で構成される.
- よって，後者のほうが一般的に早い.
- 早さはデータバスの幅が倍になっていることだけでなく，マシン語自体の洗練による貢献も大きい.
- アドレス幅の倍増は早さには関係ない.
  - そもそも，実際の64bit CPUでは64本アドレスバスが実装されていない.

386のレジスタ

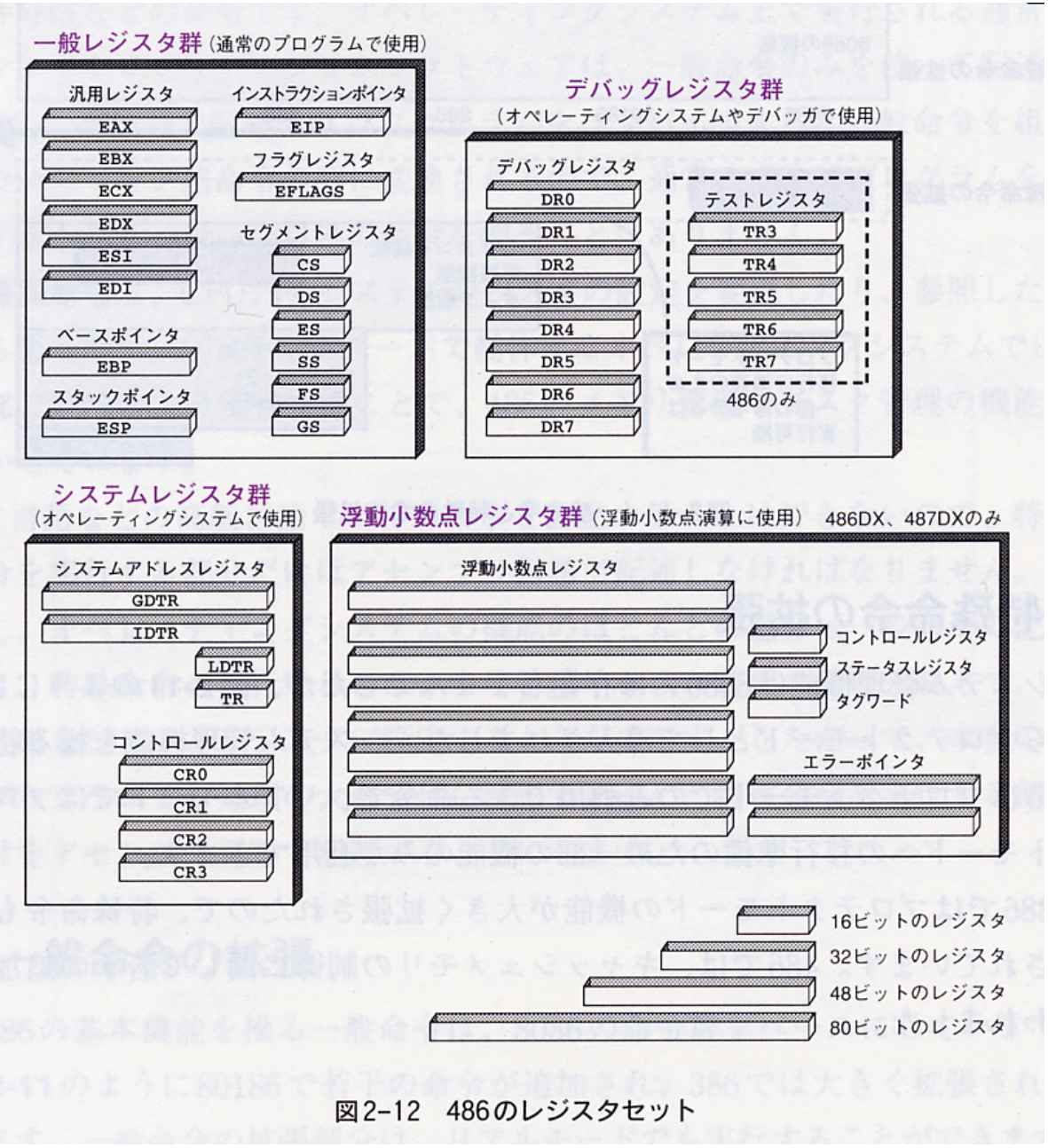
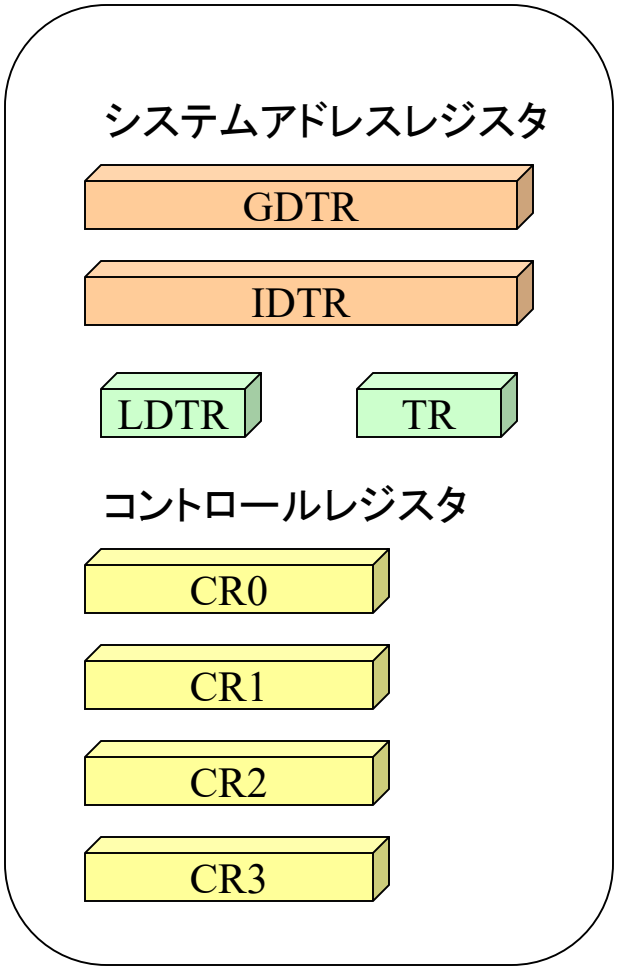
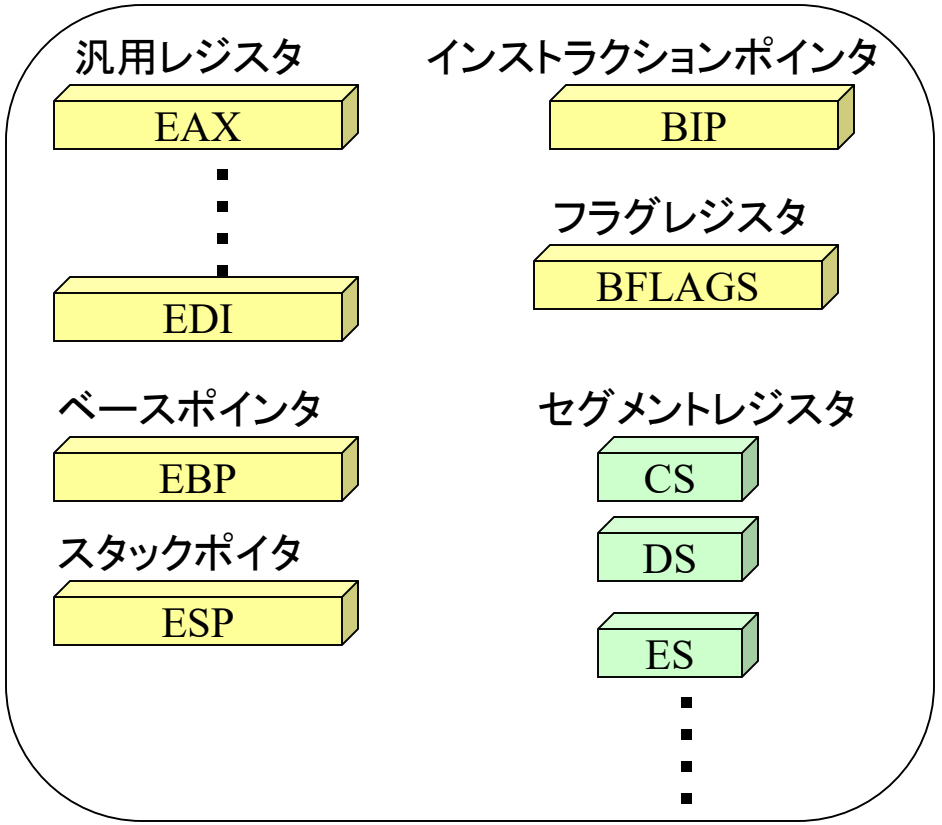


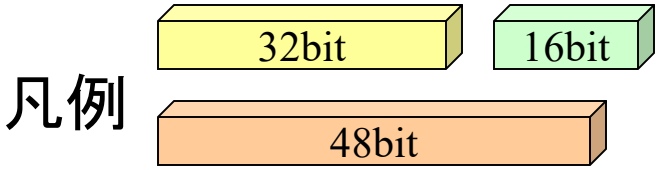
図2-12　486のレジスタセット

文献6 p.52

386のOS関係のレジスタ







1つのアプリと1つのOS

一つのOSと一つのアプリが同居する

- 複数プロセスを考える前に単一プロセスとOSの二つが動作の仕組みを理解する.

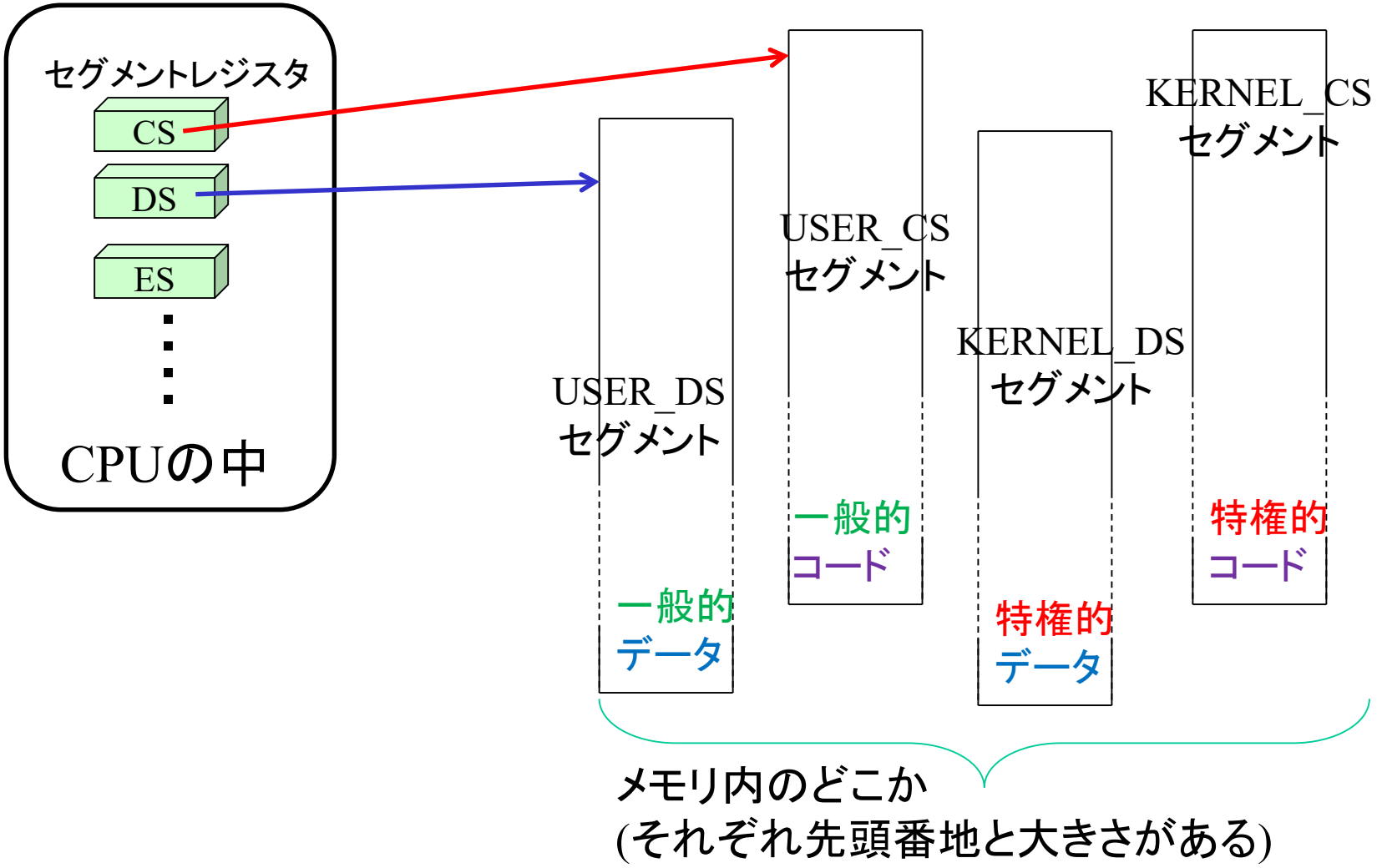
- OSのほうがアプリのプロセスより，特権的なことができなければならない.
  - システムコールのところで必然性は語りました.
- OSとCPUがどのような機能を出し合って，このような区別(特権もしくは通常)を実現しているかを説明する.

386CPUとOSの機能概要

- CPUはメモリの一部をセグメントと呼ばれる部分に区切る機能を持つ.
- CPUは現在実行中のプロセスの,
  - CSレジスタ: プログラムがあるセグメント
  - DSレジスタ: データを置くセグメント

  をレジスタに記録している.
- セグメント毎に付与されたフラグを見てCPUは実行権限の区別をしてくれる, 主なフラグは以下.
  - DPL(特権レベル): 特権命令を実行できるか否か？
    - 正確にはDescriptor Privilege Level
  - タイプ: 中身がプログラムかデータか？
- いくつのセグメントがメモリ中のどこにあり, それぞれどんなフラグを持つかを, OSが記録しておく.
- CS, DSレジスタの中身はCPU動作中に特権命令を利用してOSが切り替えられる.

ある瞬間のセグメントとレジスタ

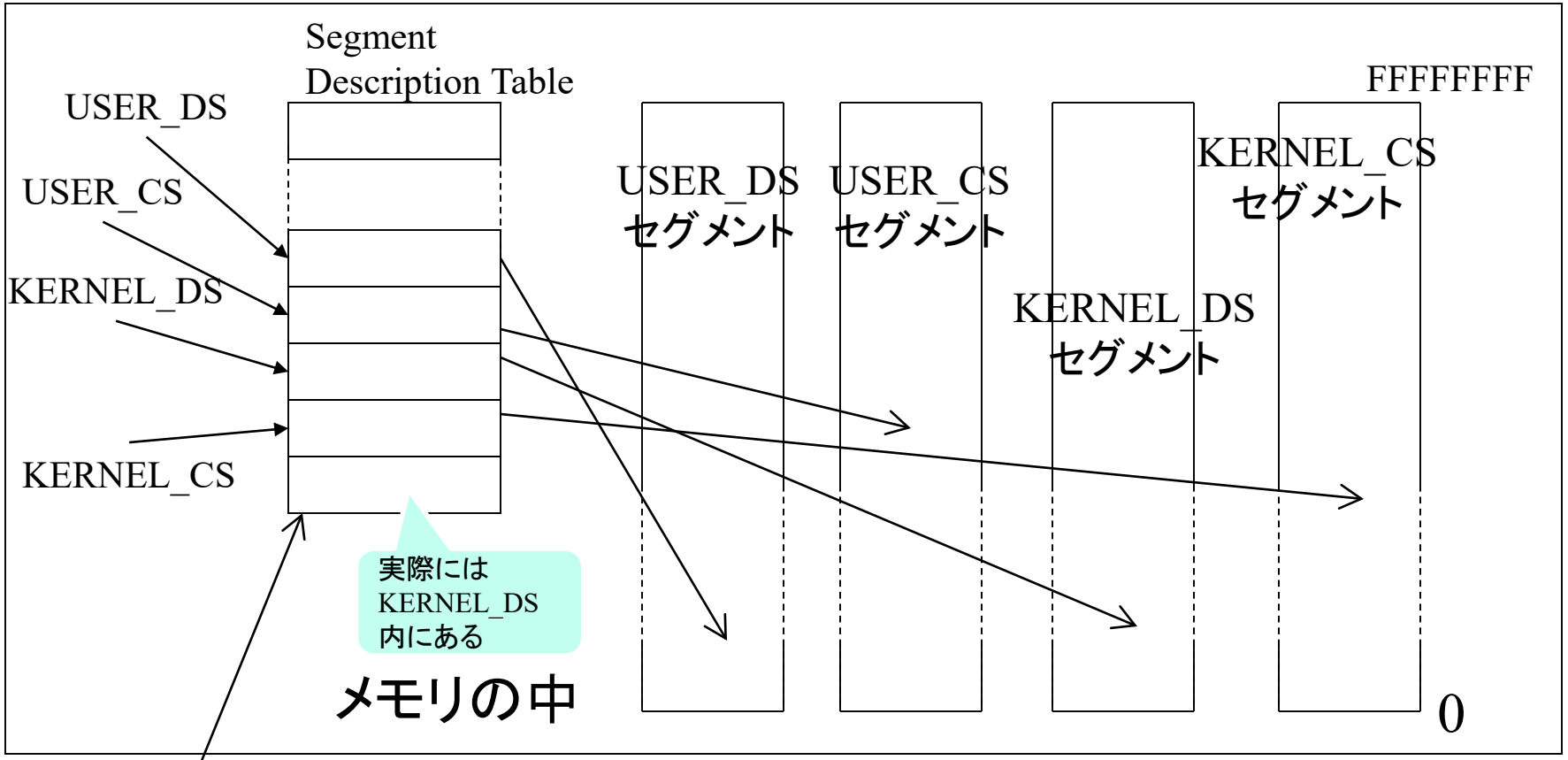


Linux(32bit v2.4)でのセグメント

- 4つのセグメントしか作らない.
  - KERNEL_CS, KERNEL_DS, UESR_CS, USER_DS
- 以下の点が共通
  - セグメントのサイズ 4G
  - セグメントの開始アドレス 0

  要は全部重なってる.
- タイプと特権レベル(DPL)について
  - *_CS タイプ5 (実行可能) 要はプログラムが入る.
  - *_DS タイプ1 (読み書き可能) 要はデータが入る.
  - KERNEL_* レベル0 なんでもできる.
  - UER_* レベル3 機能が制限.

Linuxでのセグメント概念図



CPUの中

GDTRレジスタ

Linux起動時点に,
GDTRの値をOSがセットし,
その先にセグメントの情報をOSが置く.

特権レベルについて

- 原則，同じレベルのCS, DSの組み合わせしか許されない.
  - KERNEL_CS + KERNEL_DS もしくは USER_CS + USER_DS
- セグメント毎に定義されたDPLとは別に現在動作中のCS特権レベルをCPL (Current Privilege Level)という名でCPUは記録している.
- システムコールでは USER_DS のまま，一時的に CSレジスタを KERNEL_CS に切り替える必要がある.
  - コレは割り込みを利用して切り替えを行う.

モード

- カーネルモード
  - OSのプロセスが使っているモード
  - システムレジスタの書き換え，CPUの停止等の特権命令を含め，すべての命令を呼べる.
- ユーザーモード
  - アプリのプロセスが使ってるモード.
  - 特権命令は使えない.

割り込みと例外

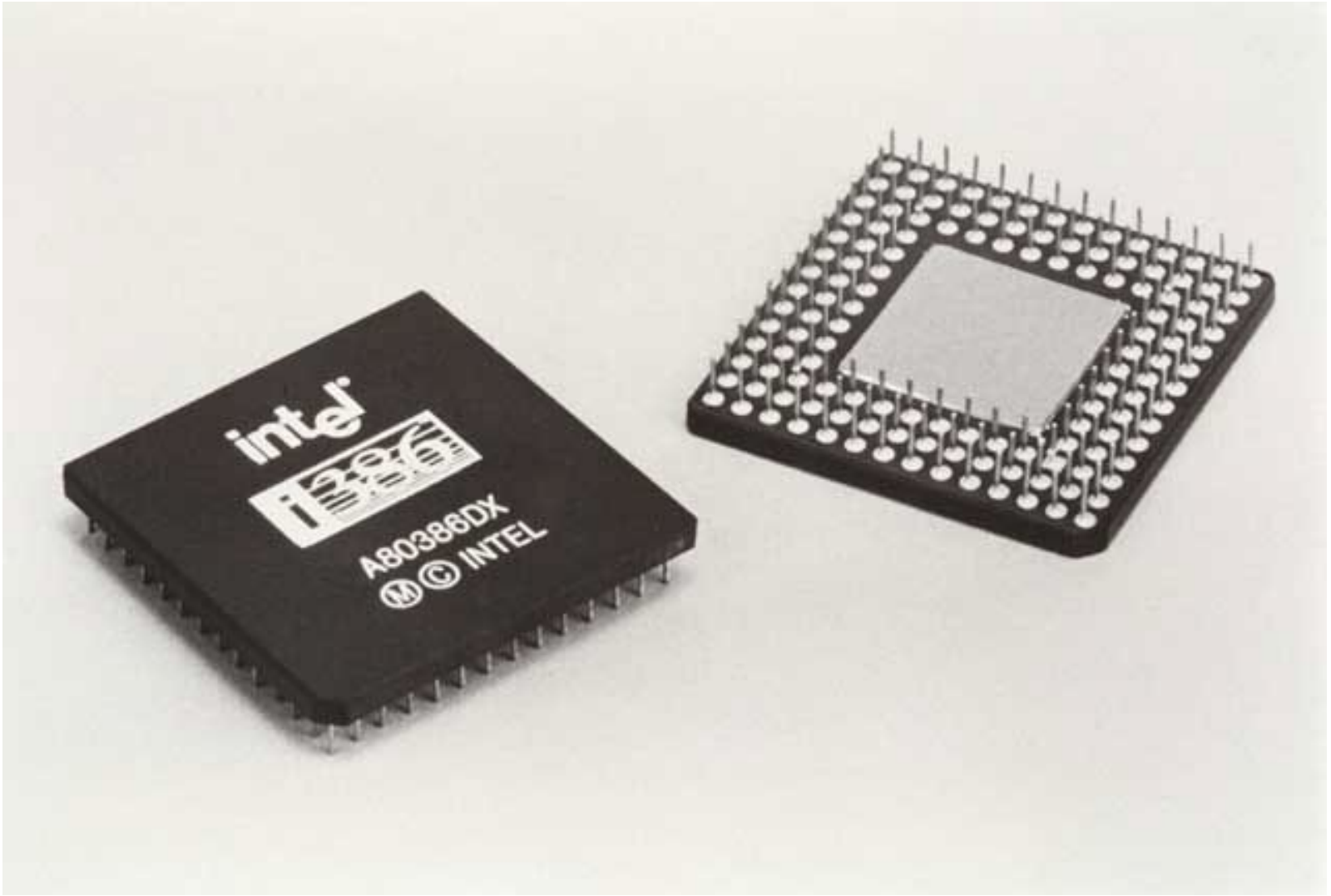
- 現在，動作しているプログラムを停止し，事前に準備した別のプログラムを実行する仕組み.
- 割り込み・例外は発生原因によって分類されている.
- 大まかにいうと，
  - 割り込み: 優先度の高い機器等からの優先処理依頼.
  - 例外: 異常事態.

  と分類できる.

i386の割り込みの発生源

- ハードウェア割り込み
  - キーボードやタイマー等の外部機器からの信号により発生. (INTRピンから受け取る)
  - メモリーエラー等により発生. (NMIピンから受け取る)
    - この割り込みのみマスク(受け取り拒否)できない.
- ソフトウェア割り込み
  - 割り込みを発生させるアセンブラ命令，INT を用いて，プログラム自身が発生させる.
    - 気分的にはサブルーチンコールに似ている.

ピン



i386例外の分類

対処法の違いにより分類

- フォルト: 割り込まれた命令を再実行する場合.
  - セグメントやページの保護違反等が原因.
- トラップ: 再実行しないもの.
  - サブルーチンコールのようなもの.
- アボート: 致命的なエラー.
  - プログラムもしくはOSの強制停止がありうる.

i386の割り込みと例外の識別

- 割り込みと例外は「割り込みベクタ」という番号によって識別される.
- 番号は 0 ～ 255 (8bit)の256個つけられる.
- 0 ～ 31 は intelが予約，32以降は自由に使ってよい.
- 通常，それぞれの割り込み番号は，

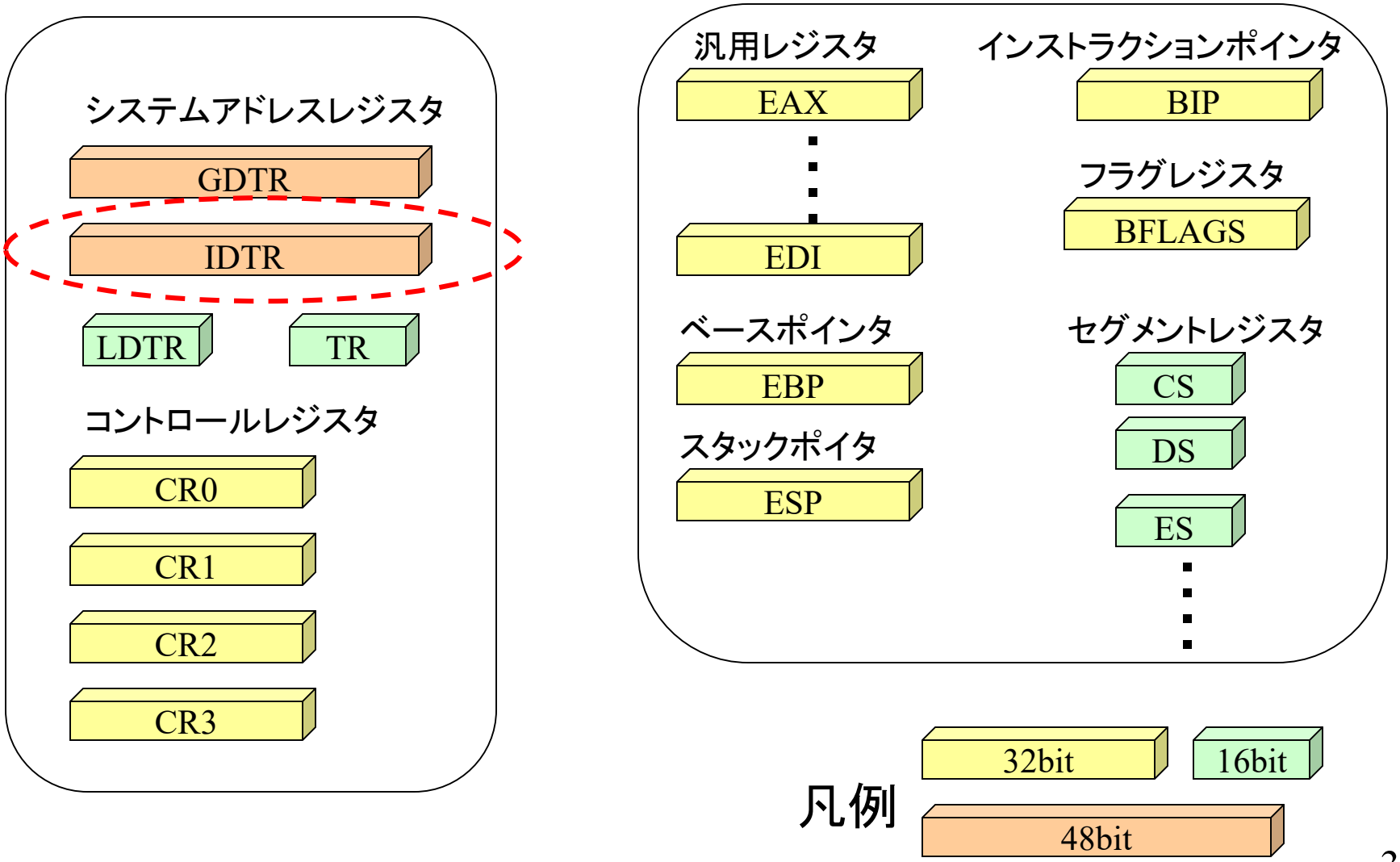
  INT X

  とかいう番号で略記される. (実際，割り込みを呼び出す場合のアセンブラ命令はこんな感じで書く)
- ベクタ番号128 (INT 0x80)は，Linuxではシステムコール実装のために使われる.

割込みディスクリプタ・テーブル

- IDT (Interrupt Description Table)
- 1個8Bのエントリを数個(≦256)列挙して，ある番号の割り込み・例外が起こったら，どのセグメントのどのコードを実行するかを定義している表.
- 表の先頭アドレスとはIDTRレジスタに記録されている.
- 後述の割込みゲートディスクリプタとトラップゲートディスクリプタの二種類のエントリがある.
- ベクトル番号(エントリの番号) 3, 4, 5, 128 だけはユーザーモードから割り込めるが，他はカーネルモードでないと利用できない.
  - 128はシステムコール実装のための利用.

i386のOS関係のレジスタ



エントリの種類

- 割込みゲート ディスクリプタ
  - この割込み実行中には他の割込みを禁止する.
  - Linuxでは割込みを全てこちらのエントリ方式で処理する.
- トラップ ゲート ディスクリプタ
  - 他の割込みを禁止しない.
  - Linuxでは例外を全てこちらのエントリ方式で処理する.
  - ユーザーモードからアクセス可能な 3, 4, 5, 128番はこちらのエントリである.

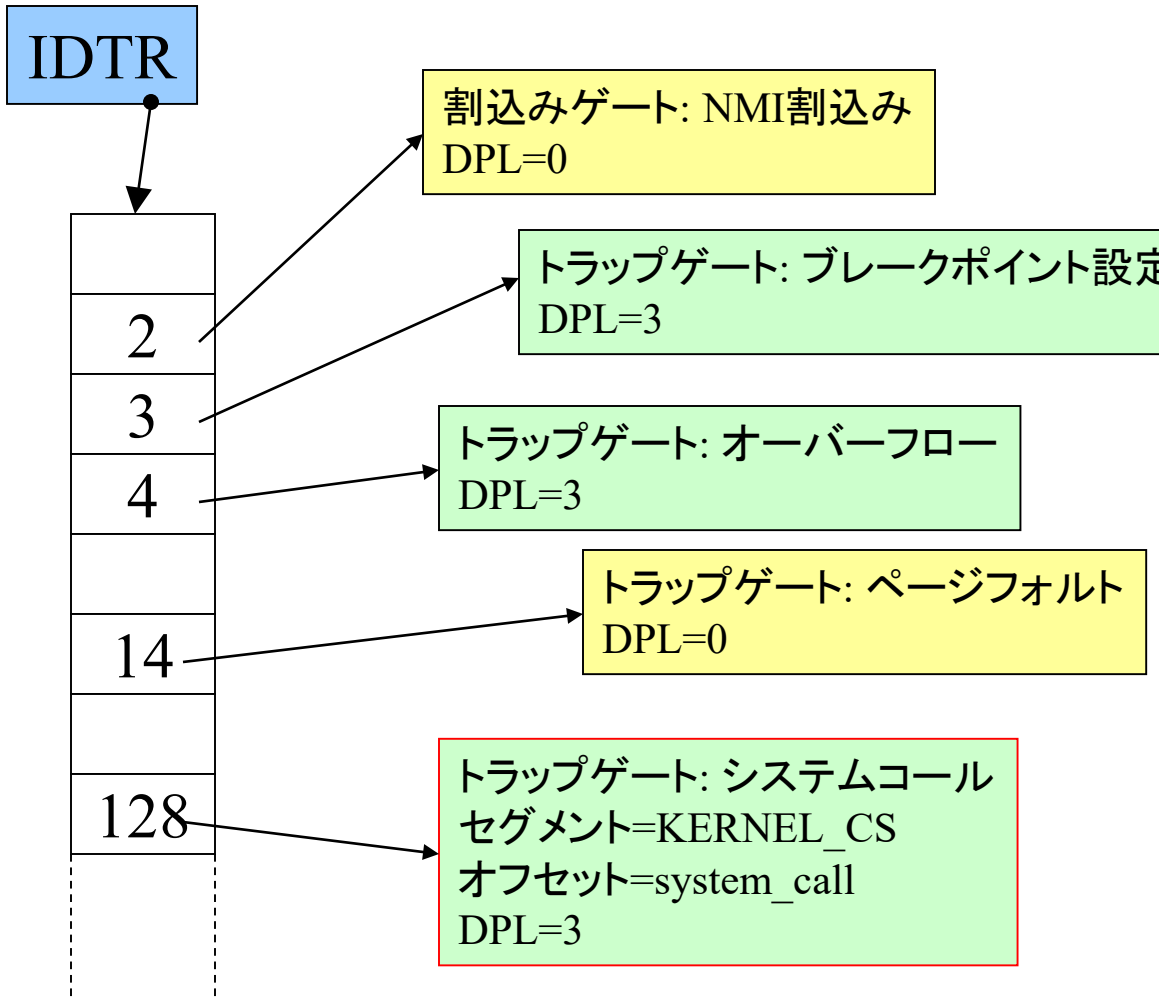
もう1種類あるけどLinuxで使わないので略.

システムコールの呼び出し

- トラップゲートはLinuxでの例外ハンドラを記述するのに使われていた.
- トラップゲートは以下のように作られている.
  - トラップゲート記述内の飛び先(ハンドラー)をKERNEL_CS内にしておけば, USER_CSから(レベル3)からでも, レベル0の機能を呼び出せる.
  - 例外ハンドリングかサブルーチンコールかの違いだけ.
- (前述のように)システムコールの呼び出しは, この「ゲート」によってi386では実現されている.

例

参考: DPL=0 特権
DPL=3 一般



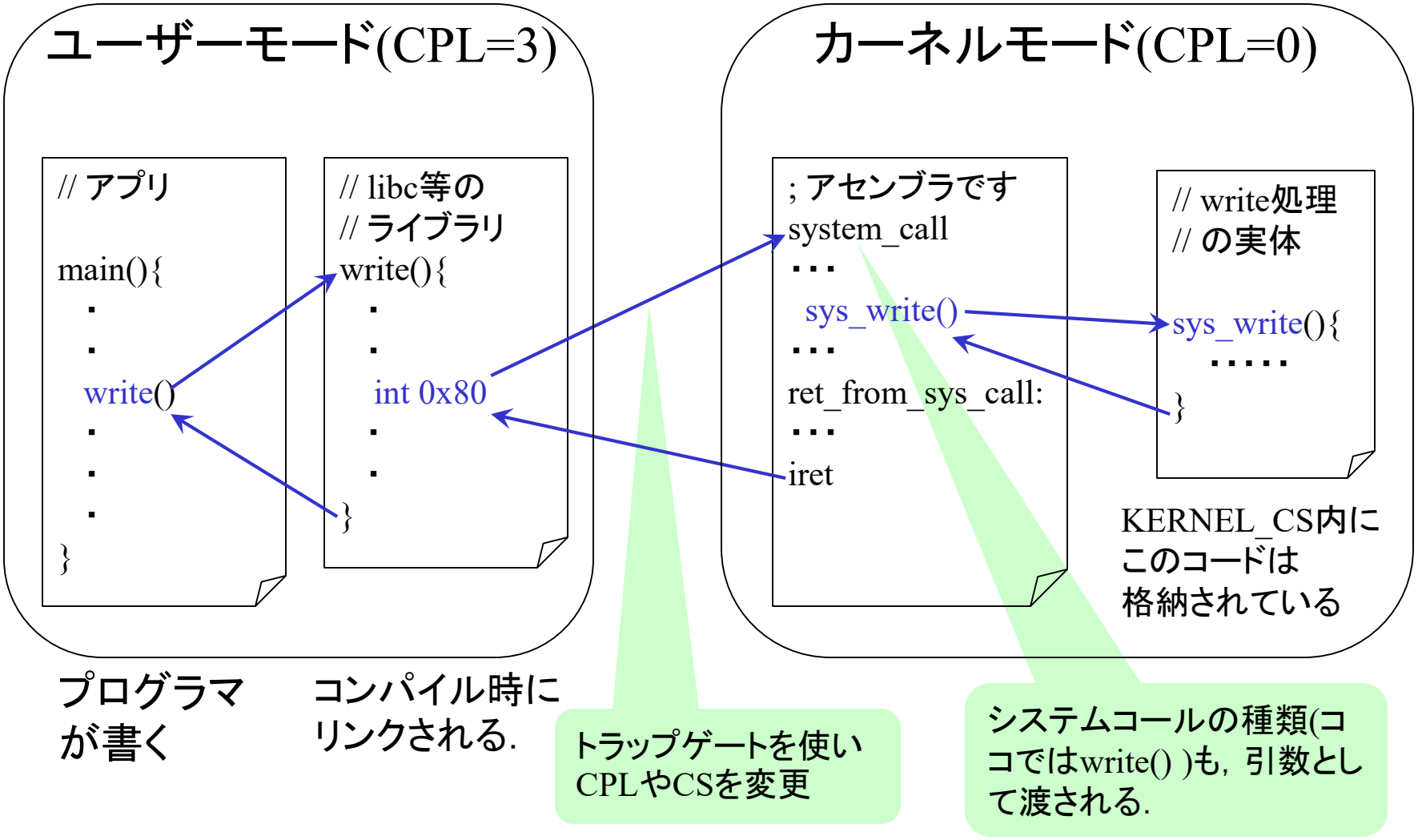
int 128

をCPL3で実行すると，制御はKERNEL_CS内のsystem_callという関数があるアドレスに分岐し，同時にCPLも0に変化する．

システムコールが呼ばれるまで

- 前述のようにトラップゲートを利用して，CPLを変化させることで，システムコールは実現されている．
- では，次項で具体的にユーザープログラム内のシステムコールの呼ばれる手順を追う．

例えばwrite()

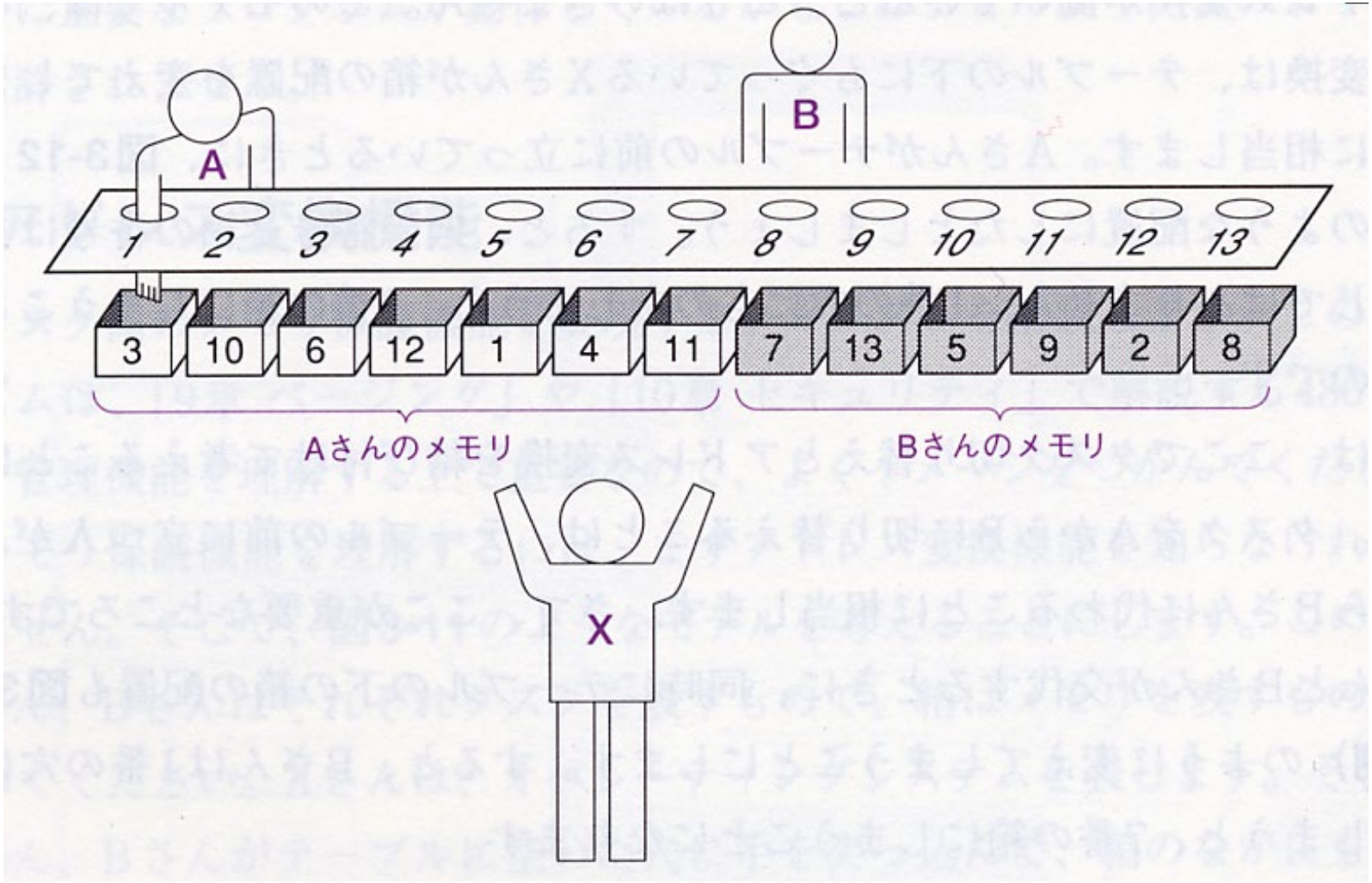

ユーザーモード(CPL=3)
カーネルモード(CPL=0)
// アプリ
main(){
write()
}
// libc等の
// ライブラリ
write(){
int 0x80
}
; アセンブラです
system_call
sys_write()
ret_from_sys_call:
iret
// write処理
// の実体
sys_write(){
}
KERNEL_CS内に
このコードは
格納されている
プログラマ
が書く
コンパイル時に
リンクされる.
トラップゲートを使い
CPLやCSを変更
システムコールの種類(コ
コではwrite())も，引数とし
て渡される.

多数のプロセスとOS

多数プロセスの交通整理: メモリ

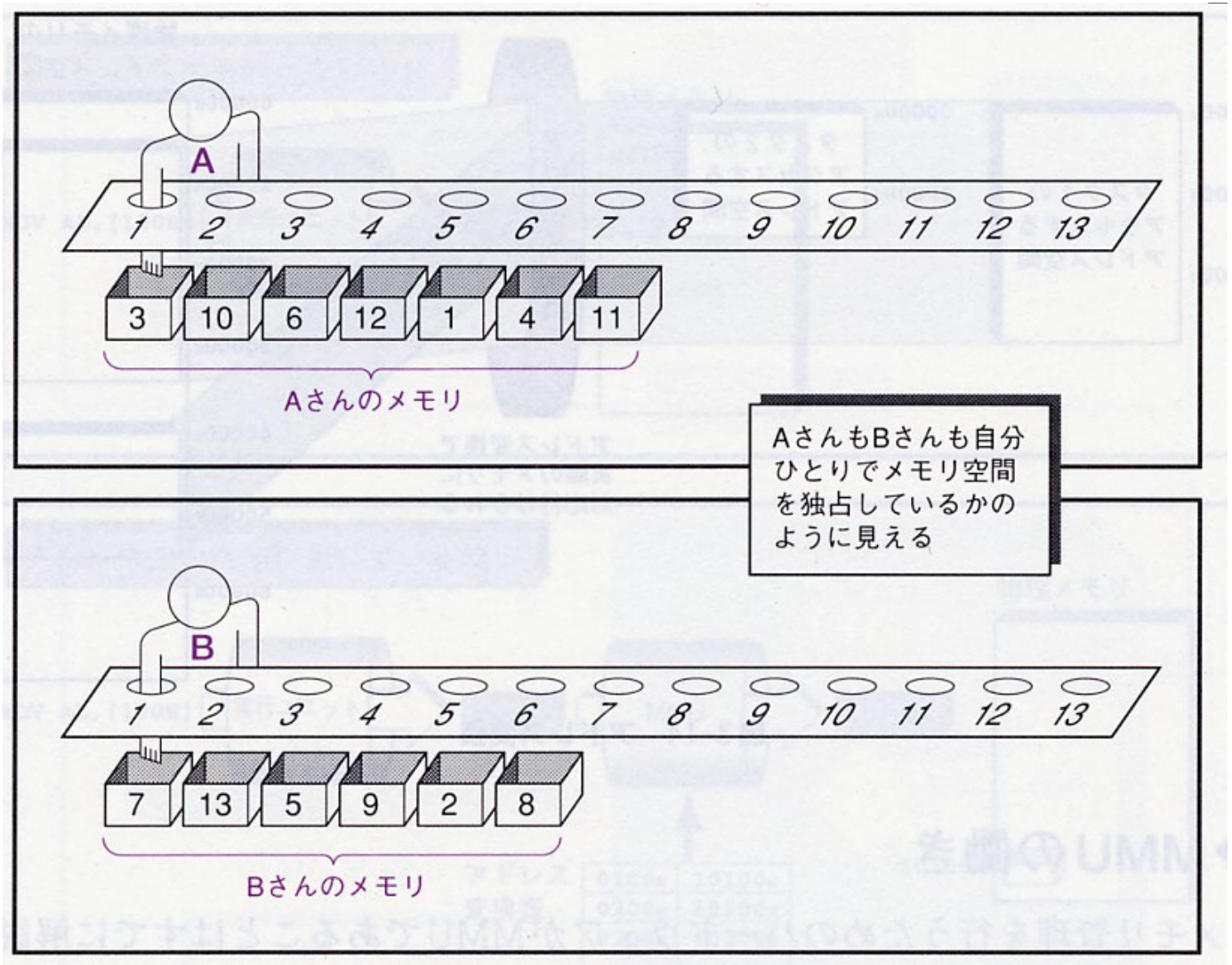
- Linuxではアプリ用のセグメント(USER_CS, USER_DS)は一対しかない.
- ご存じの通り200個近いプロセスが同時に存在するのが普通.
- 異なるプロセス同士が同じメモリを使ったら混乱が生じるのは前回話した通り.
- Linuxではアドレス変換テーブルを切り替えることで，複数のプロセスが同じアドレスをアクセスしても，実際には違うメモリ回路を使うように工夫されている.

アドレス変換の考え方 1



文献6 p.79

アドレス変換の考え方 2



ページング

- 0～4GBで表現されるリニアアドレスを物理アドレス(実際にマシンに搭載されたアドレス)に変換する仕組み.
- 4GBも実際積んでないマシンでも4Gあるように振る舞えるのはこの機構のおかげ.
- もし4GBメモリのマシンがあったとしても, i386上で，複数のプロセスが同時に動作するOSを稼動させるには必要な技術.

ページ

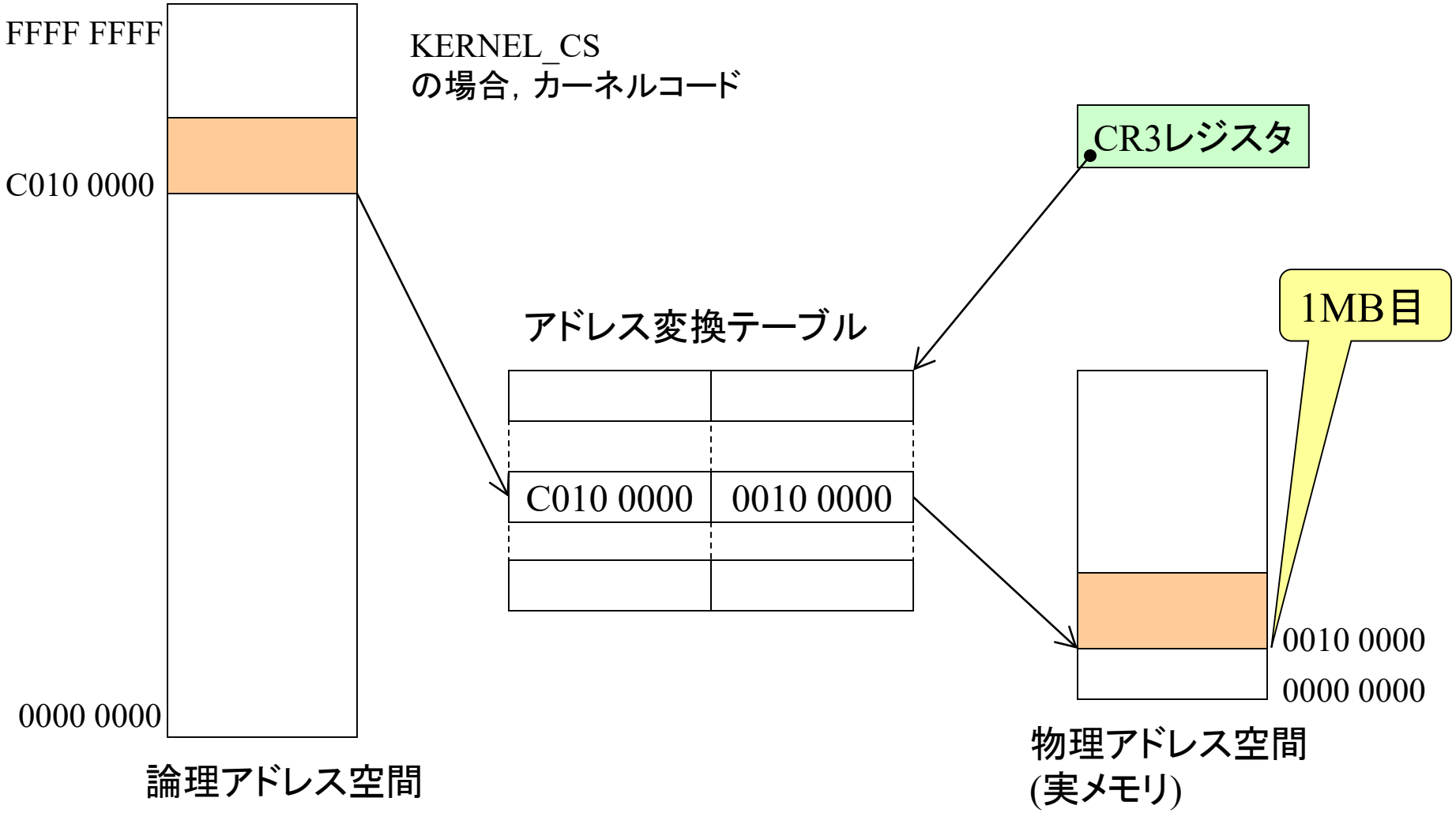
- セグメントを4KB(もしくは4MB)の固定長に分割した個々の部分のこと.
- リニアアドレスのある部分が物理アドレスのどの部分に対応するかは表で管理している.

  ⇒ アドレス変換テーブル

アドレス変換テーブル

- 前述のようにリニアアドレスのある部分が物理アドレスのどの部分に対応するかを記述した表.
- 無論，メモリ内に記入される.
- ある瞬間に利用されているテーブルは1つだが，変更もできる.
- i386ではCR3レジスタにこのテーブルがあるアドレスが記入されている.

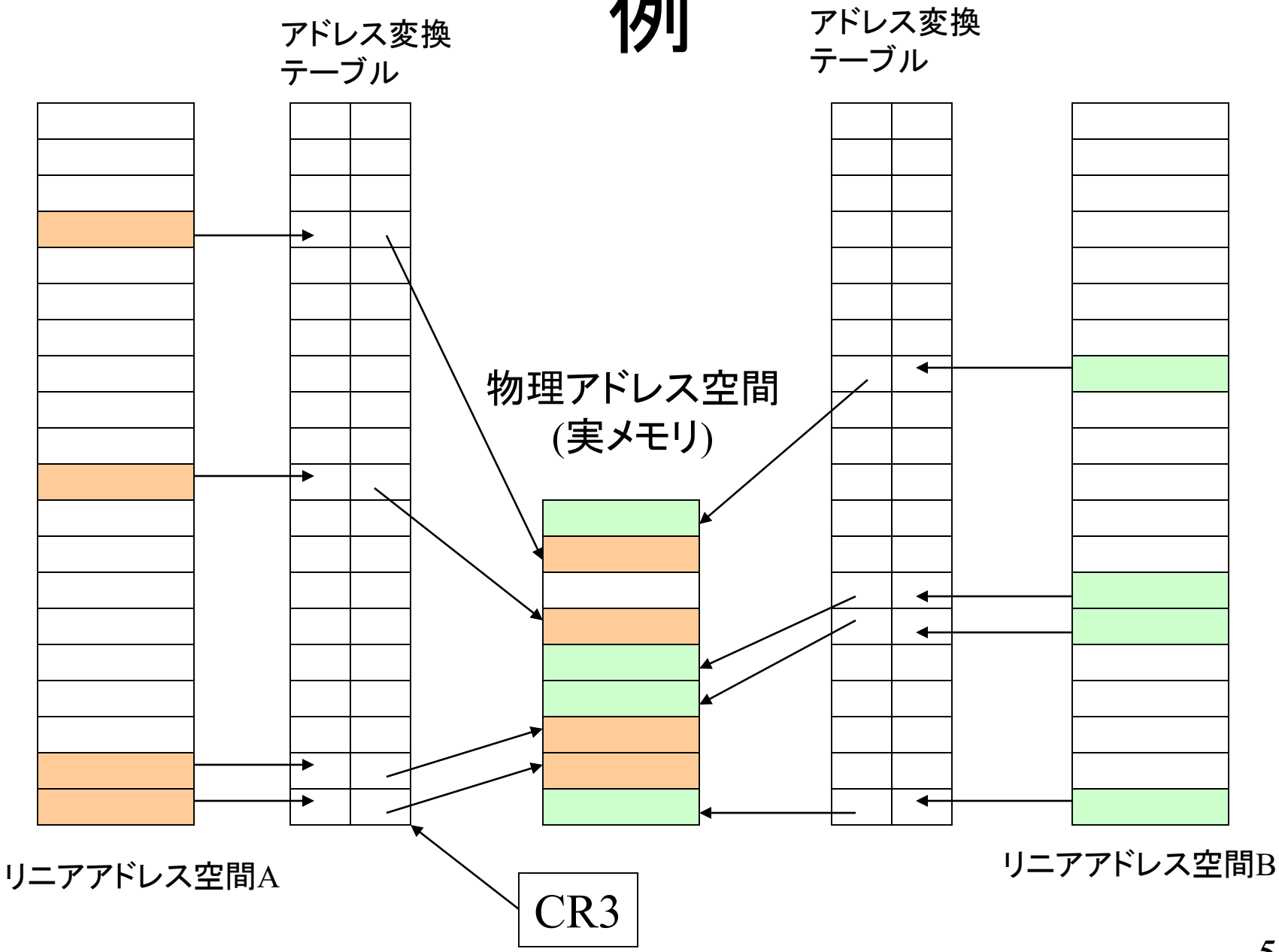
アドレス変換テーブルの例



アドレス変換テーブルとプロセス

- Linuxではプロセス毎にアドレス変換テーブルを作る.
- さらに，プロセス間で同じ物理アドレスを使わないようにテーブル内の値をOSが設定する.
  - 意図的に共有させることもできる.
- ユーザープロセスがこのテーブル群やCR3の内容は変更できないので安心.
  - テーブル群はKERNEL_DSに書かれる.
  - CR3はKERNEL_CS内のコードでしか変更できない.
- 実際のテーブルは効率化のため，二段階テーブルになっているが，それについては後日に.

例


アドレス変換
テーブル
アドレス変換
テーブル
物理アドレス空間
(実メモリ)
リニアアドレス空間A
リニアアドレス空間B
CR3

i386の3つのアドレス記述法

- 論理アドレス: マシン語でのアドレス指定に使う形式．セグメントとオフセットの対で表現．
- リニアアドレス: 4GBのメモリ空間を素直に表現した形式．32ビット
- 物理アドレス: メモリチップの実アドレス．表現は32ビットだが，上限は実際に搭載しているメモリ量に依存．

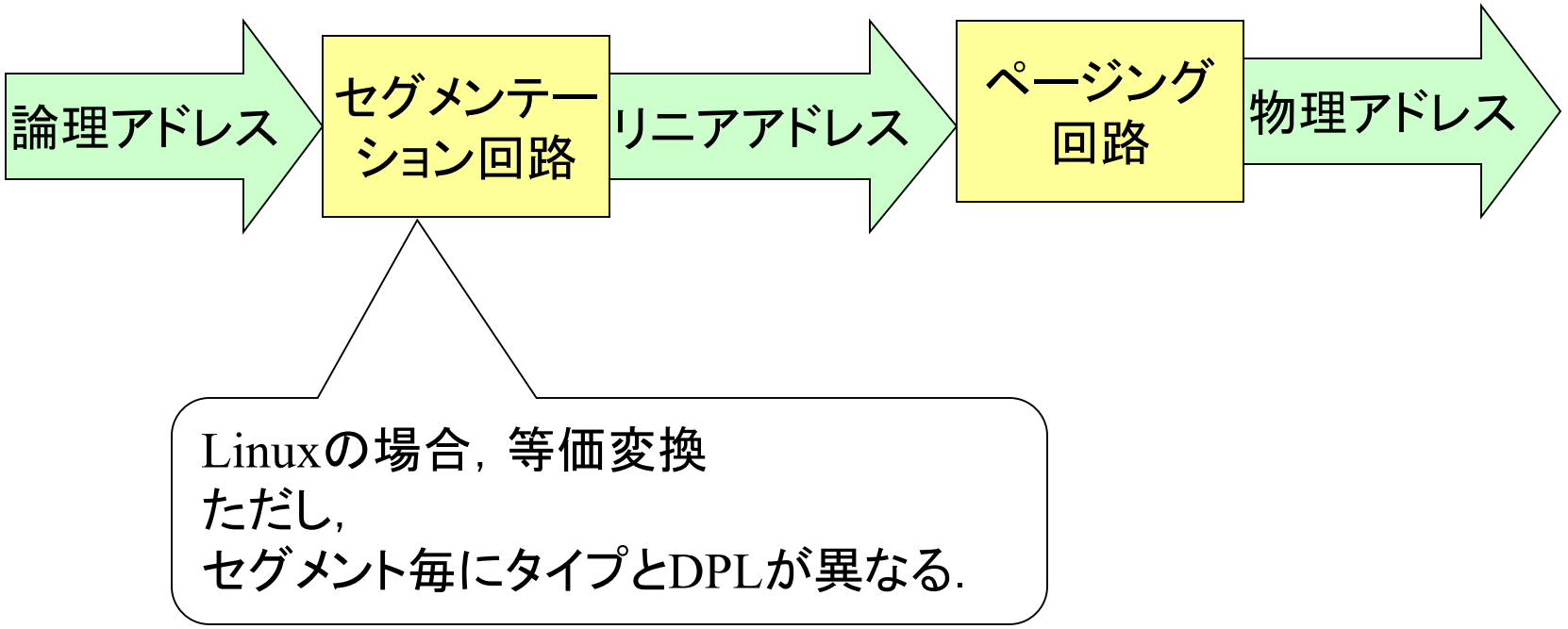
Linuxの場合，4Gサイズのセグメントを使うので，

論理アドレス = リニアアドレス

文献5 p.44

アドレス変換

マシン語が解釈されて，実際のメモリ回路までたどり着くには，以下のような二段階の変換が行われる．



多数プロセスの交通整理: 他

- コード，データセグメントだけでなく，個々のプロセスが計算を行うには，レジスタが必要.
  - 386ではTSS(Task State Segment)と呼ばれる情報を出し入れして，レジスタの中身を切り替えるようです.
- 入出力装置について
  - アプリのプロセスが装置を利用の際，OSに利用申請を出して使うので，交通整理ができてます.

プロセス切り替え方式・今昔

- I/Oイベント駆動型
  - 実行中のプロセスが入出力に入ったら，他のプロセスに実行権を移す手法.
  - あるプロセスの事情が他のプロセスに影響を与えるため，「完全なマルチプロセスでは無い」といわれる.
  - 実現が簡単な古い方法.
  - Windows3.1で採用されていた，古い型の大型機でも.
  - 「マルチプログラミング」と呼ばれることあった（プログラミングとは無関係だけど）.
- プリエンプティブ型
  - Preemptive
  - 各プロセスの事情に関係なく，一定間隔でプロセスを切り替える.
  - CPUのタイマーが大きな役割を果たす.
  - 技術的に実現は上記より難しい.
  - TSSはこちらの技術を使っている.

再掲載

まえおき: カーネルの役割

- カーネルはアプリの実行を支援する

- 個々のアプリ
  - アプリが必要な資源をカーネルが供給する.
  - カーネルがアプリ実行を開始し，終了する.
- 多数のアプリ
  - カーネルがアプリ群の交通整理をする.
    - 必要な資源を排他的に共同利用
    - それぞれスムーズに動く

単一プロセスの管理

- プロセスの実体は,
  - プログラムを読み込んだCS
  - 計算結果等を読み書きするDS
  - レジスタの中身 (主に一般用)
  - アドレス変換テーブル

  と言える.
- OSはファイルから読み込んだプログラムに基づき, これらを生成・設定する.
- OSは全プロセスの上記情報を覚えておく.
- より具体的なプロセス生成, 終了の手順は後日に.

複数プロセスの交通整理

- プロセスの切り替えタイミングはタイマー任せでランダムに行う.
- レジスタの中身は専用の命令で切り替える.
- メモリの中身はアドレス変換テーブルで, プロセス同士がかち合わないようにする.
- 入出力装置はOSが一元管理して, 利用がかち合わないようにする.

本日は以上

アンケートのほう，
よろしくご提出ください