システムソフトウェア

2021年度

第3回 (10/11)

月曜7-8限・木曜7-8限(Zoom)

講義担当：渡部卓雄 (Takuo Watanabe)

http://titech-os.github.io

e-mail: takuo∅c.titech.ac.jp

本日のメニュー

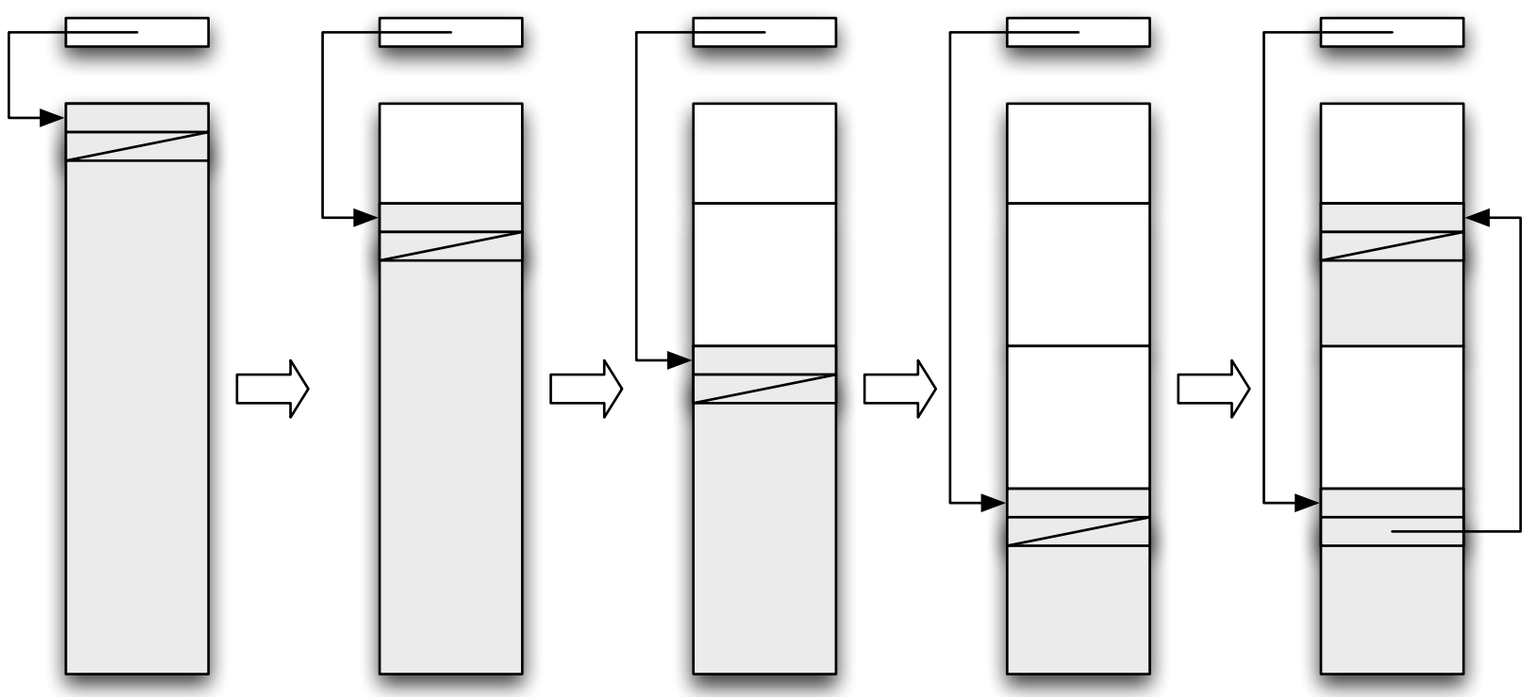
- メモリ管理(1)

メモリ管理の目的

- 抽象化
  - メモリの割当や解放を自動化したい
- 仮想化
  - 各プロセスがあたかもコンピュータのメモリを占有しているかのようにふるまえるようにしたい
  - 物理メモリより大きなメモリ空間を扱いたい
- 保護
  - 各プロセスのメモリ領域は，他のプロセスから勝手に読み書きされないようにしたい

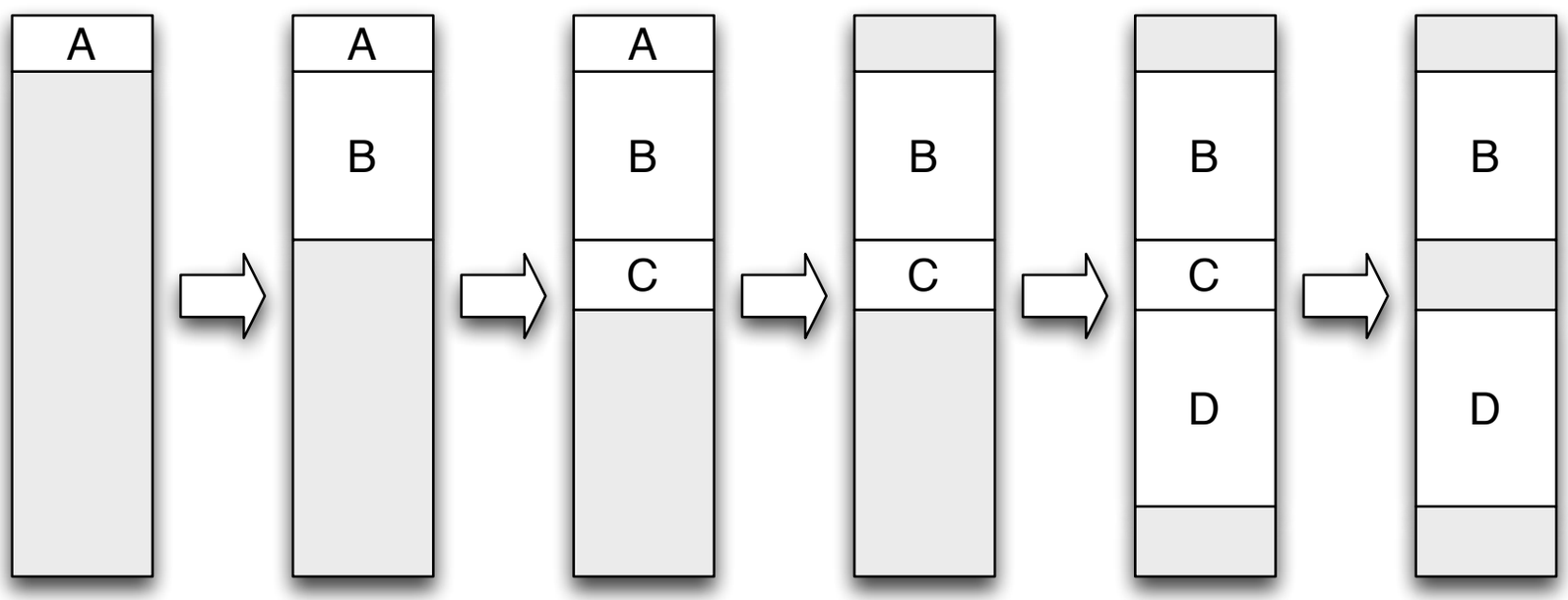
素朴なメモリ管理手法

- 必要な領域を順に割当て，空き領域を線形リスト等で管理する.



断片化(fragmentation)

- メモリの割当と解放をつづけているうちに，小さな空き領域ばかりができる現象.
  - 空き領域の合計は大きくても，まとまった領域がとれなくなる.



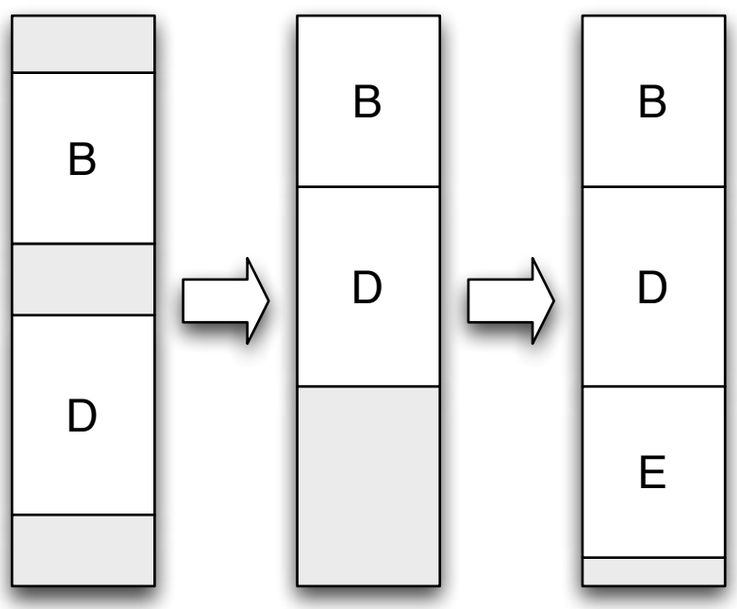
様々な割当て手法

- first-fit
  - 大きさSの領域が要求されたとき，最初に見つけた大きさS以上の空き領域から割り当てる
- best-fit
  - 大きさS以上で最も小さいものから割り当てる
- worst-fit
  - もっとも大きな領域から割り当てる

- どの方式でも断片化は同様に生じてしまう

コンパクション(compaction)

- 断片化した空き領域がまとまるよう，割当済みの領域の位置を変えること.
  - 再配置(relocation)の手間がかかる
  - アドレスの変化に対応する必要がある
- cf. 旧Mac OSのハンドル



素朴なメモリ管理いろいろ

- 可変パーティション方式(variable partitioning)
- 固定パーティション方式(fixed partitioning)
  - メモリを固定サイズのパーティション(ブロック)単位で割り当てる.
  - 必要なメモリサイズをM, ブロックサイズをB, ブロック数をnとすると, $(n-1)B < M \leq nB$. このとき $nB - M$ だけの余分な空きが生じる. この空きによる断片化を内部断片化と呼ぶ.

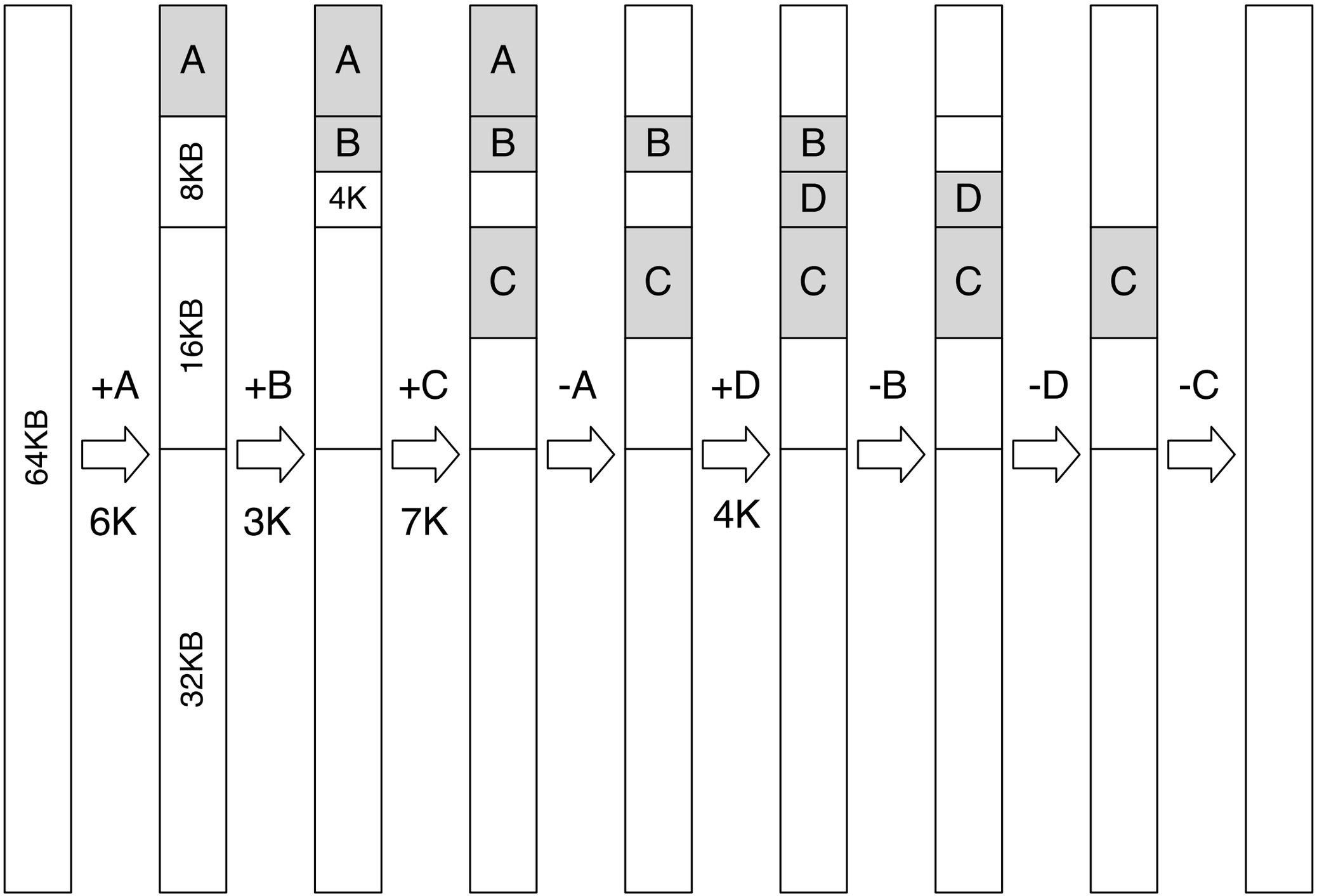
空き領域の管理

- リスト
  - 線形リスト，2重連結リスト
  - 大きさ毎のリスト
- ビットマップ
- バディシステム(buddy system)
  - Linuxカーネル等で用いられている

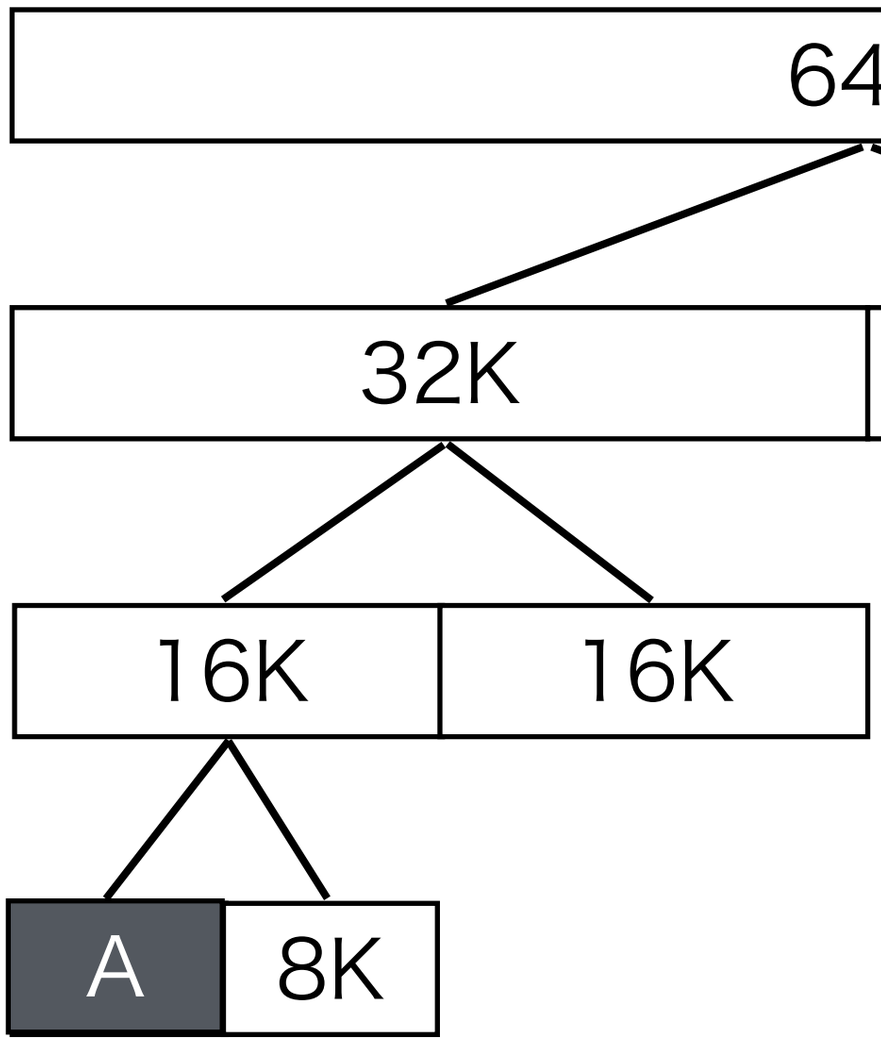
バディシステム

- 物理的に連続するページで構成される固定長セグメントからメモリを割り当てる方式
- 2の巾乗アロケータ(power of 2 allocator)により，4KB, 8KB, 16KB, ... 等の単位で要求に応える.
- 割当可能なメモリを割当に必要な最小の単位になるまで2個ずつのペア(buddy)に分割していく.
- メモリブロックが解放されたとき，ペアの両方が空いていれば合体(coalescing)を行い，空き領域を大きくする.

バディシステム



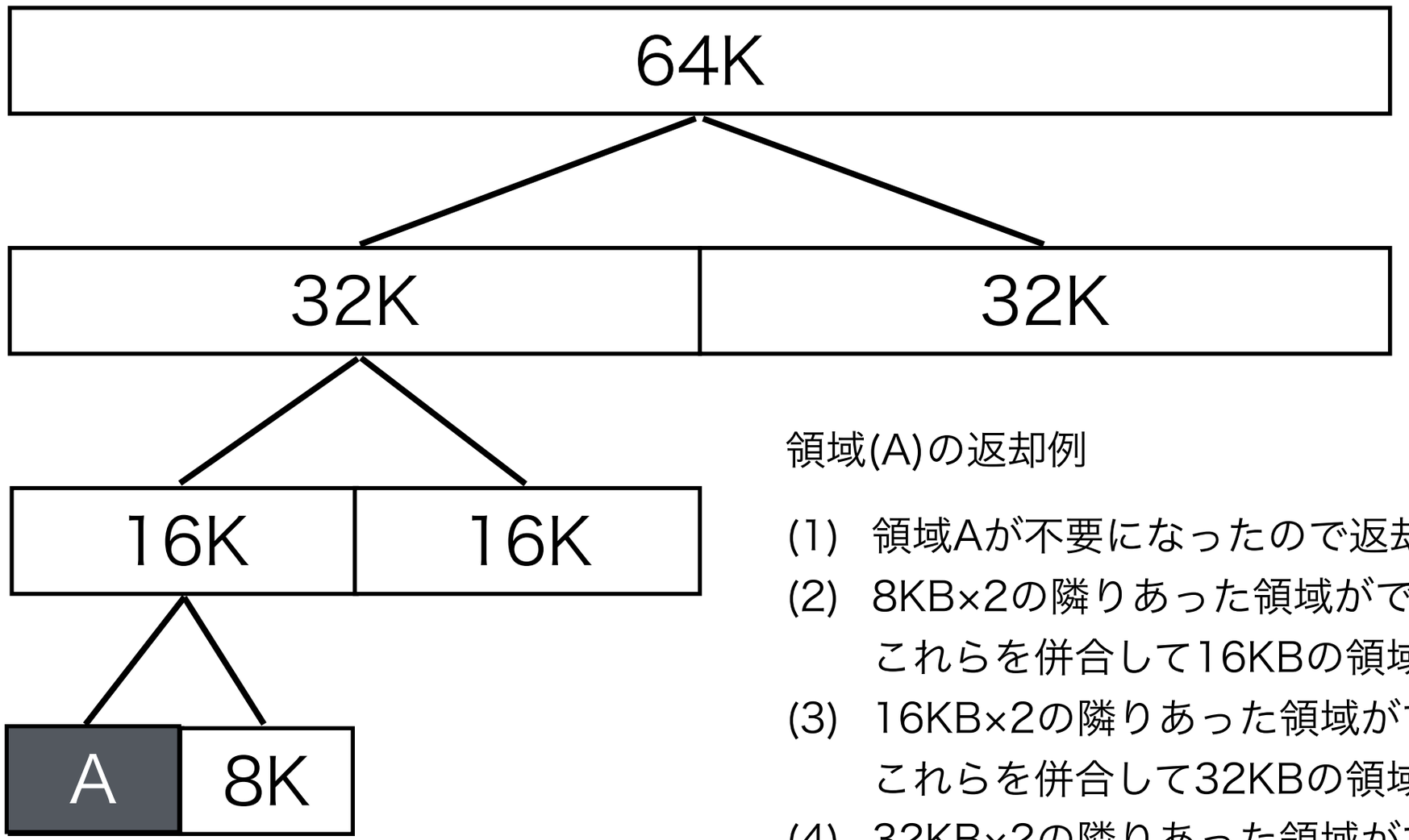
バディシステムでの領域割り当て



6KBの領域(A)の割り当て例

(1) 64KBでは大きすぎるので32KB×2に分割
(2) 32KBでは大きすぎるので16KB×2に分割
(3) 16KBでは大きすぎるので8KB×2に分割
(4) これ以上分割すると要求されたメモリサイズ未満になってしまうので，8KBのバディの片方をAとして割り当てる.

バディシステムでの領域返却



領域(A)の返却例

(1) 領域Aが不要になったので返却
(2) 8KB×2の隣りあった領域ができたので,
これらを併合して16KBの領域にする
(3) 16KB×2の隣りあった領域ができたので,
これらを併合して32KBの領域にする
(4) 32KB×2の隣りあった領域ができたので,
これらを併合して64KBの領域にする

素朴なメモリ管理の問題点

- メモリ保護がない.
  - プロセス同士の干渉が起こり得る.
- 物理メモリ量以上の割当てができない.

アドレス空間(address space)

- 論理(logical)アドレス空間
  - プロセスが利用しているアドレス空間
- 物理(physical)アドレス空間
  - コンピュータ上のハードウェアとして実装されているメモリのアドレス空間

アドレス変換(address translation)

- 論理アドレスを物理アドレスに写像すること
  - 通常CPU内部に搭載されているメモリ管理ユニット(memory management unit, MMU)によっておこなわれる.
  - MMUは論理アドレスと物理アドレス間の写像を表として保持している. この表をページテーブルと呼ぶ.

ページテーブル(1)

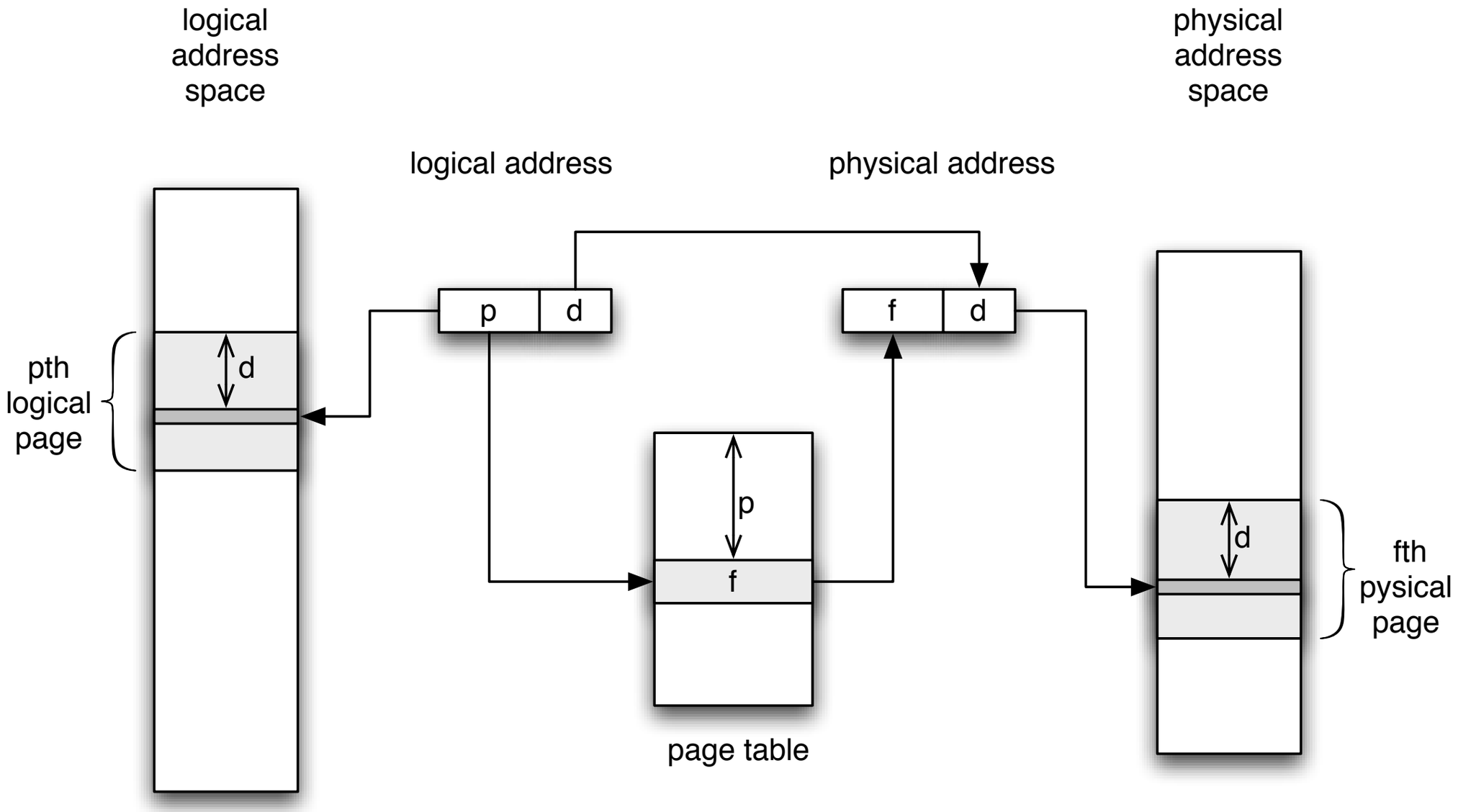
- 論理アドレスから物理アドレスへの写像をどうやって実現するか
  - 素朴に表をつくるとメモリ空間と同じサイズに…
- ページ
  - アドレス空間上の固定した大きさの領域
    - 論理アドレス空間上：論理ページ
    - 物理アドレス空間上：物理ページ（ページフレーム）
  - 4KB, 8KB程度
- MMUはページ単位での写像を行う.

ページテーブル(2)

- 論理アドレス

- 論理アドレスへのアクセスが生じると，MMU はページ番号をキーとしてページテーブルを参照し，対応する物理ページを得る.
- ページテーブル
  - ページ間の写像を実現する表
  - メモリ上に作成される

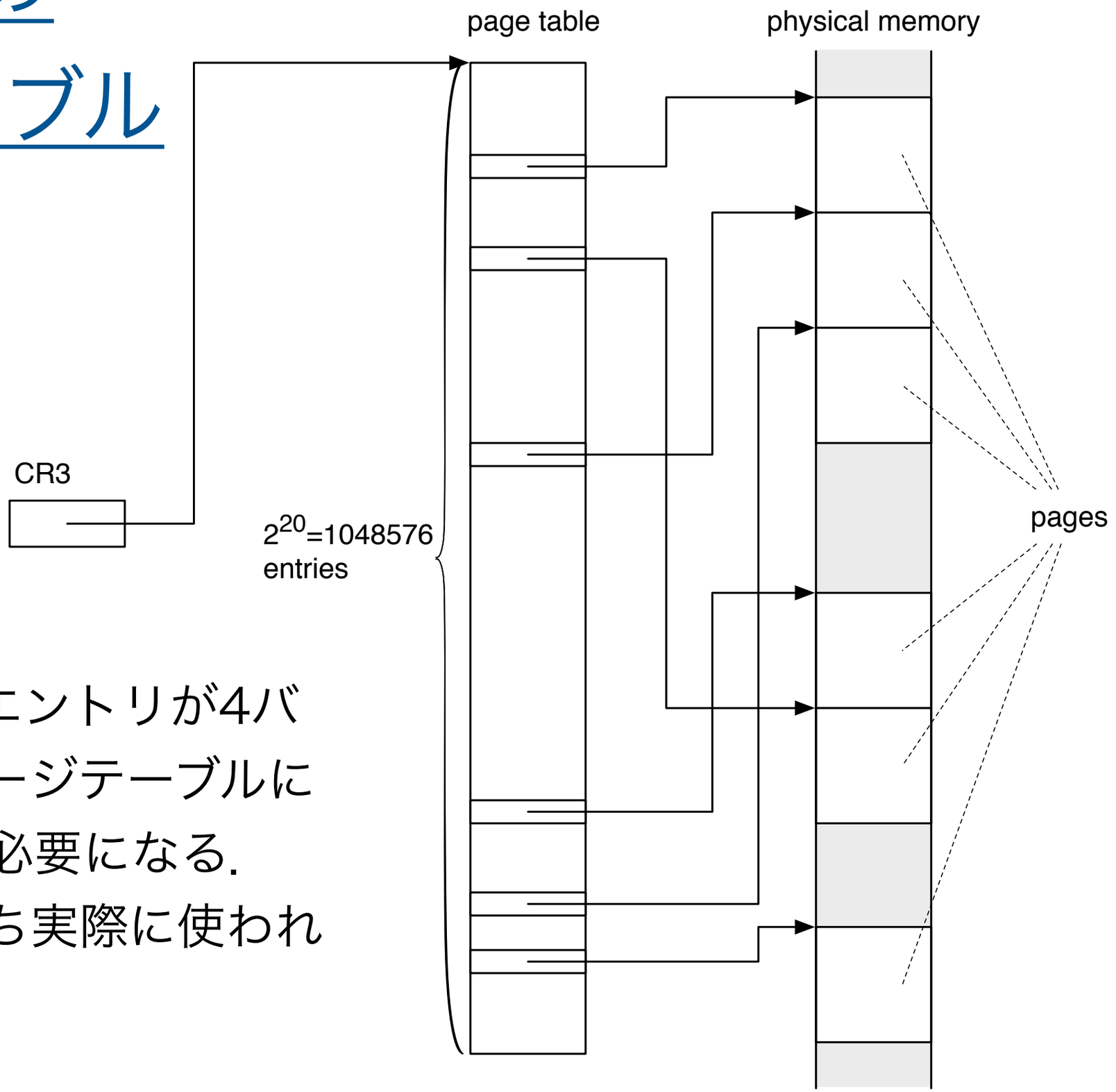
ページテーブル(3)


logical address space
physical address space
logical address
physical address
p
d
f
d
pth logical page
d
p
f
d
fth pysical page
page table

ページテーブルの大きさ

- 32ビットの論理アドレス空間で大きさ4KBのページを扱う場合，ページ番号は20ビット必要になる．この場合のページテーブルの大きさは2^{20}になる．
- 各プロセスが使うメモリは論理アドレス空間のごく一部であることが多い

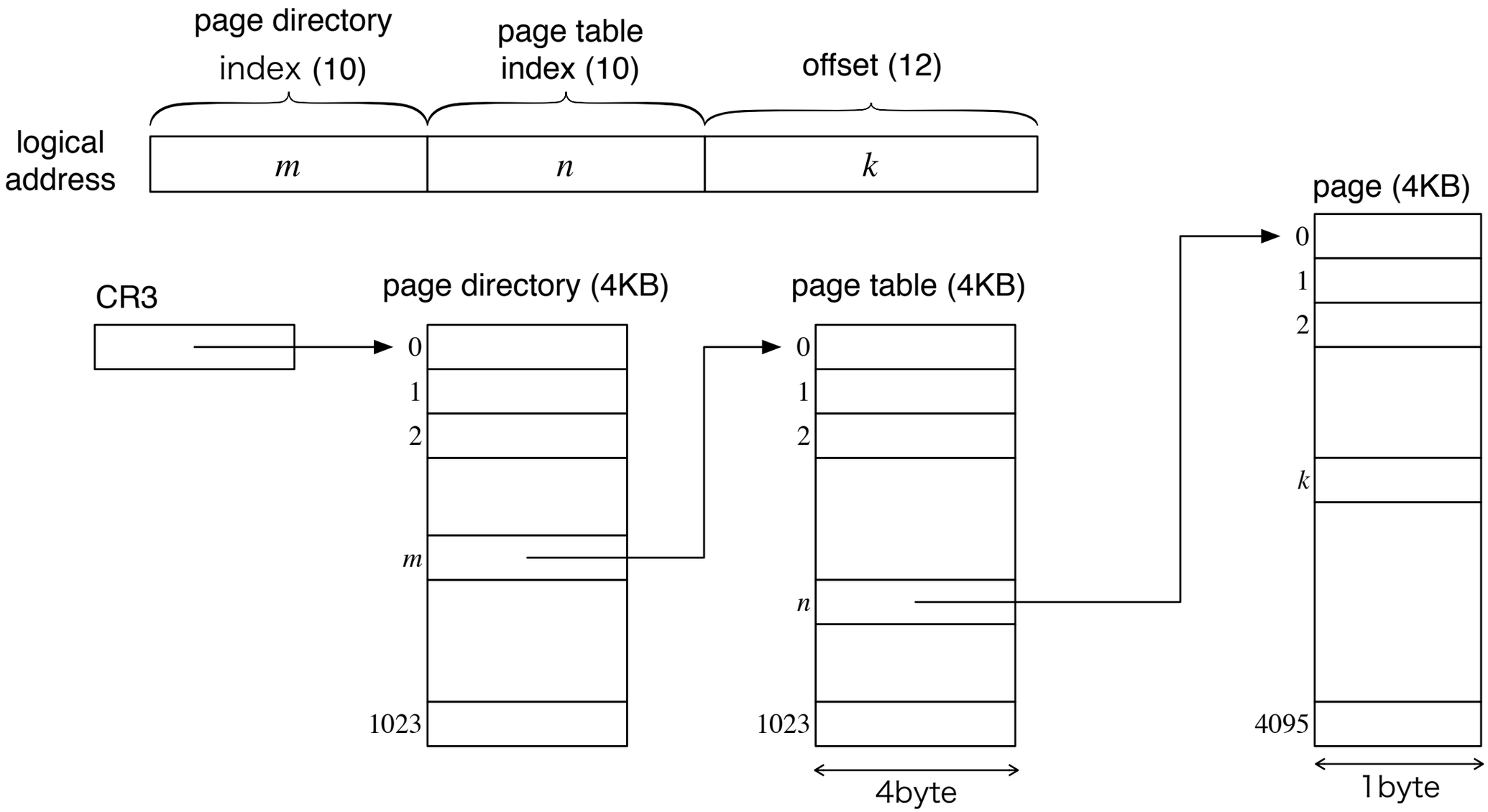
大きさ2^{20}のページテーブル



ページテーブルのエントリが4バイトとすると，ページテーブルに$2^{20}\times4$=4Mバイト必要になる.
しかしこれらのうち実際に使われるのは一部である.

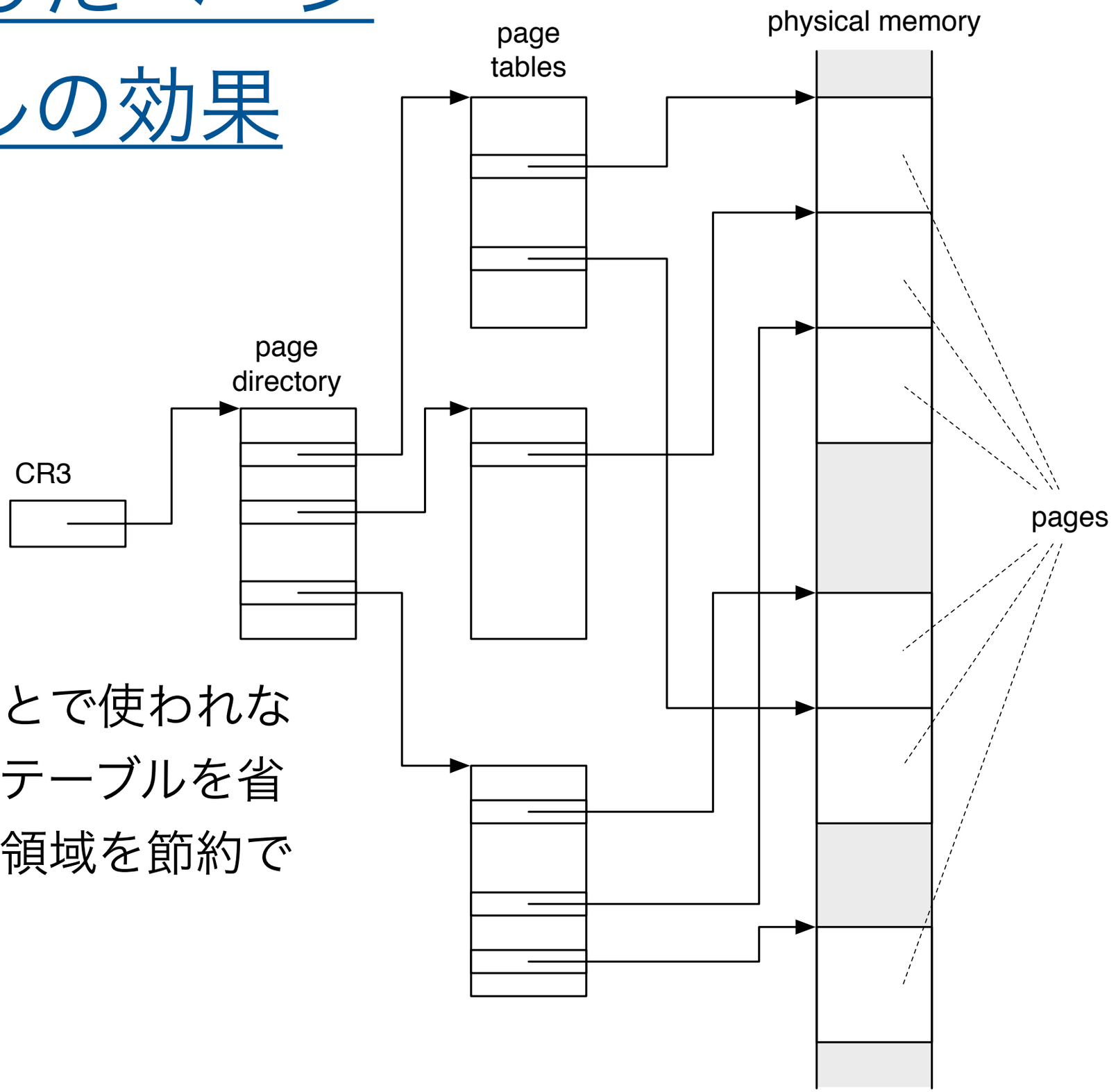
多段ページテーブル

x86(IA-32)のページテーブル



- ページディレクトリおよびページテーブルのエントリ数は1024で，大きさはページと同じ4KB.
- プロセスが切り替わる際にCR3レジスタの内容を変更することで，プロセス毎に異なるメモリ空間を実現する.

多段化したページテーブルの効果

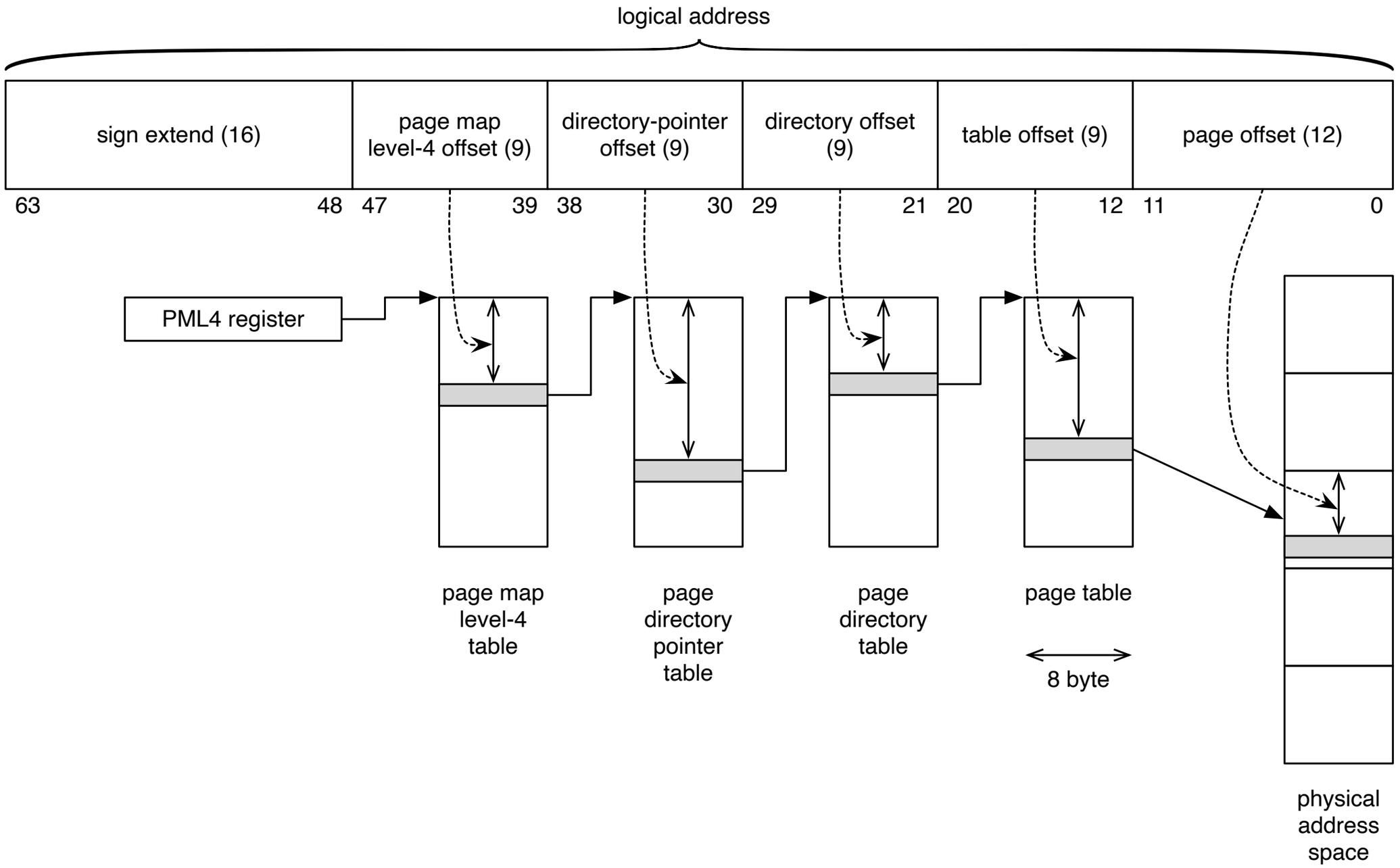


多段にすることで使われない分のページテーブルを省略でき，記憶領域を節約できる.

64ビットのアドレス空間

- 2^{64}B = 18446744073709551616B
  = 16EB（エクサバイト）
- 参考
  - Google: 10EB (2013年時点での非公式試算)
  - Dropbox: 1EB (2018)
  - TSUBAME 3.0
    - メモリ：計135TB（256GB x 540ノード）
    - ストレージ：約17PB（Lustre + スクラッチ領域）
  - 京
    - メモリ：計1.26PB（16GB x 82944ノード）
    - ストレージ：30PB

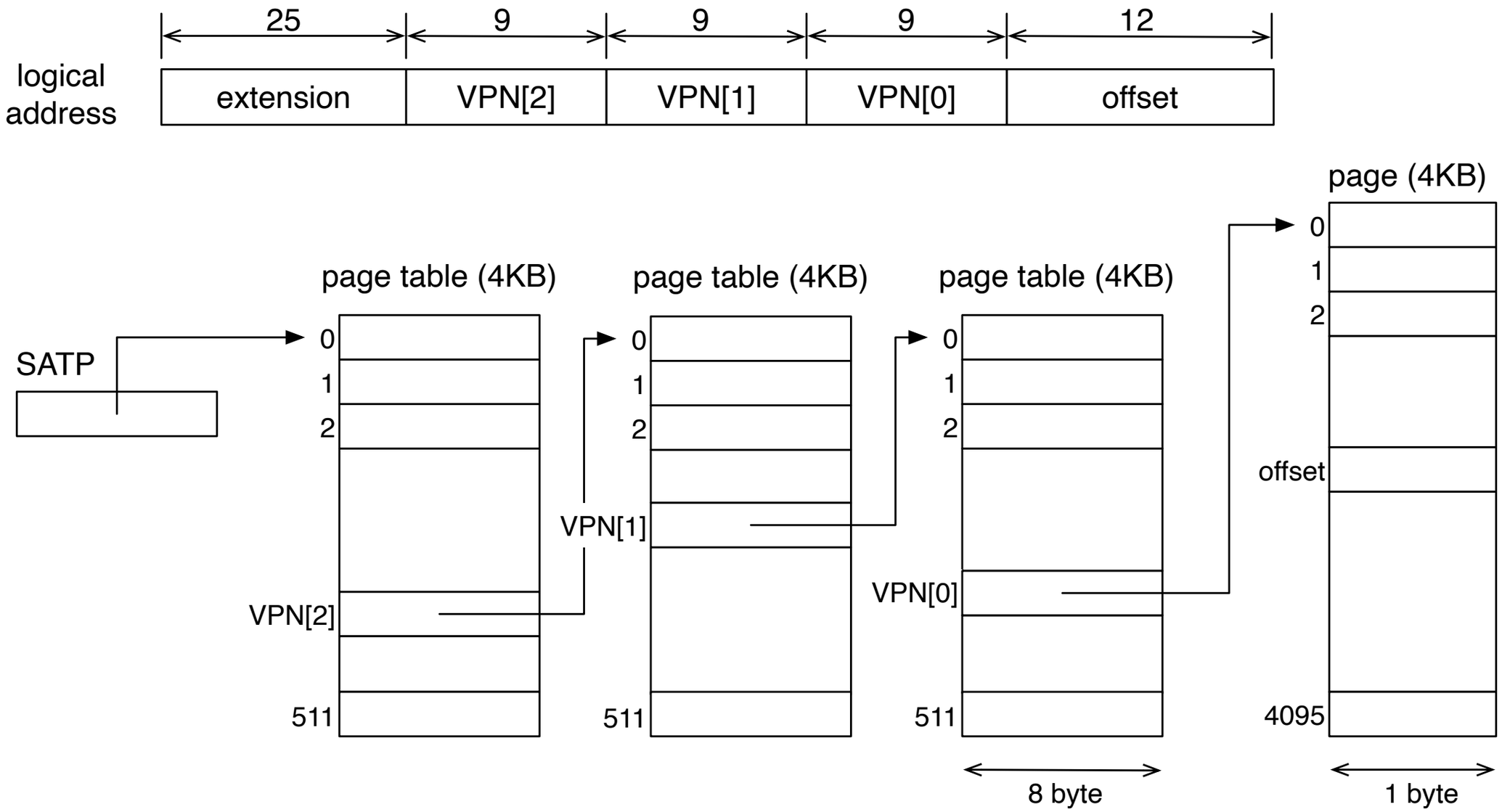
AMD64のページング



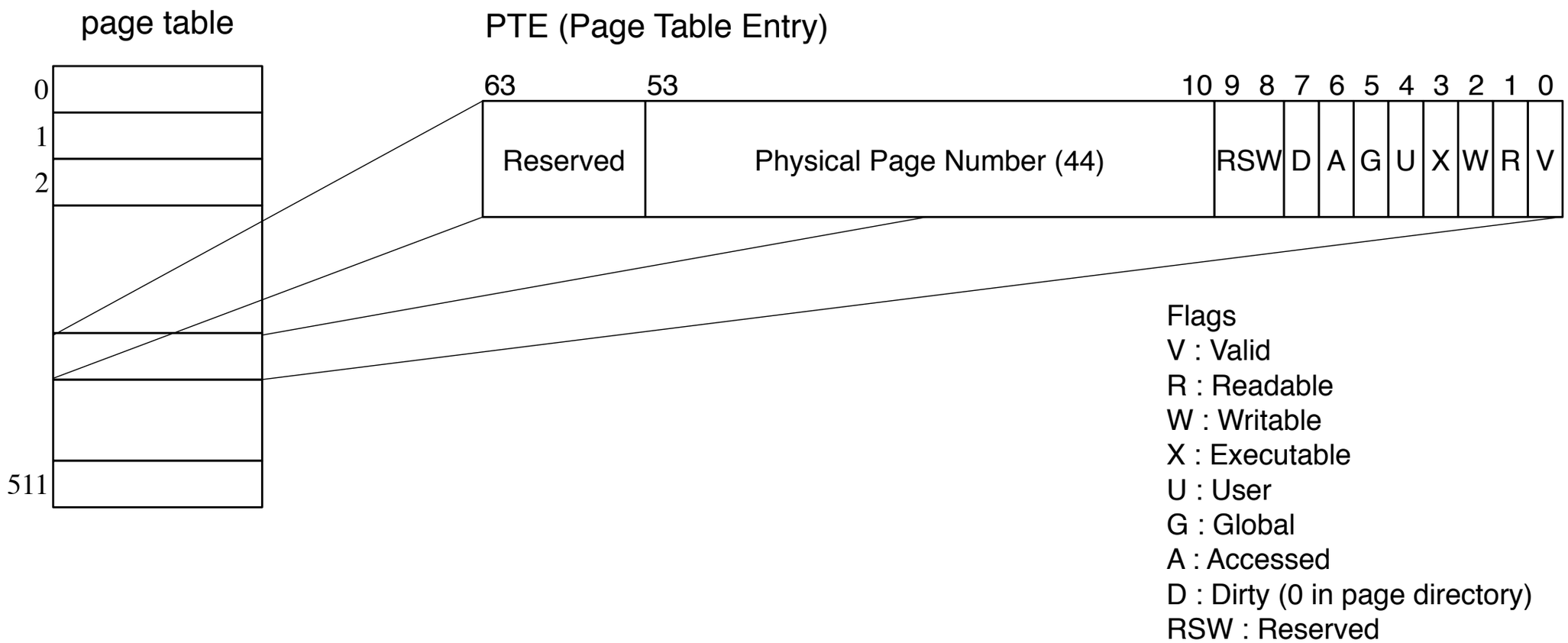
RISC-Vのメモリ空間

- RV32：32ビット版
  - 論理アドレス空間：Sv32 (4GB)
- RV64：64ビット版
  - 論理アドレス空間
    - Sv39 : 39ビット(512GB) xv6はこちらを採用
    - Sv48 : 48ビット(256TB)
  - 物理アドレス空間：56ビット(64PB)

RISC-V (RV64, Sv39)のページング



RISC-VのPTE


page table
0
1
2
511
PTE (Page Table Entry)
63
53
10
9
8
7
6
5
4
3
2
1
0
Reserved
Physical Page Number (44)
RSW
D
A
G
U
X
W
R
V
Flags
V : Valid
R : Readable
W : Writable
X : Executable
U : User
G : Global
A : Accessed
D : Dirty (0 in page directory)
RSW : Reserved

TLB(translation lookaside buffer)

- ページテーブルはメモリ内にあるため，参照にはメモリアクセスが必要になる.
- それによる性能低下をさけるため，ページテーブルのキャッシュをCPU内部に設ける．このキャッシュをTLBと呼ぶ.
  - ページ番号による検索を高速にするため，記憶内容でアクセスできる特殊なメモリ(content addressable memory)で構成される.
    - 連想メモリ(association memory)とも呼ばれる.

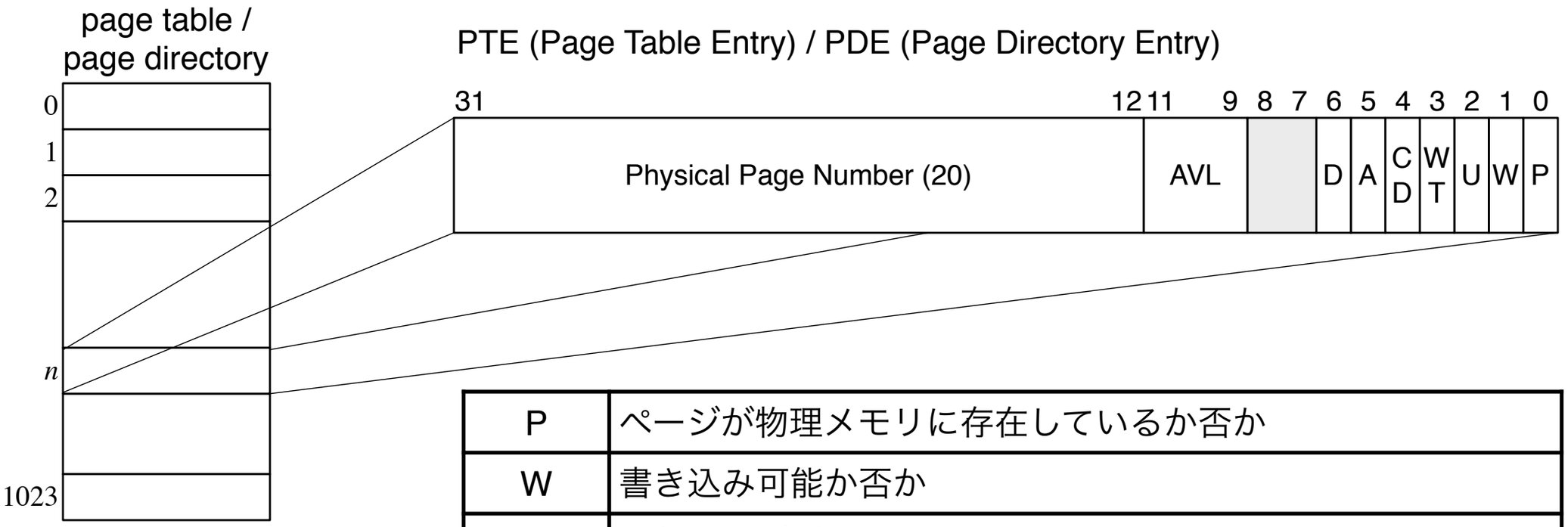
ページフォルト

- 多くのプロセスは，論理アドレス空間のすべてを使うわけではない．したがってプロセスが使わないメモリ領域は物理アドレスに写像する必要がない．
- ただし，物理アドレスに写像されていない論理アドレスへのアクセスを行うと割り込みが発生するようにしておく．
  - ページフォルト(page fault)

ページの保護

- ページテーブルの各エントリには，物理ページのアドレス(番号)の他に，ページ毎の属性値が格納されている.
  - 属性値：読み出し・書き込み・実行の許可等
- ページ属性に違反するアクセスがおこると割り込みが生じる.
  - メモリ保護違反(protection fault)

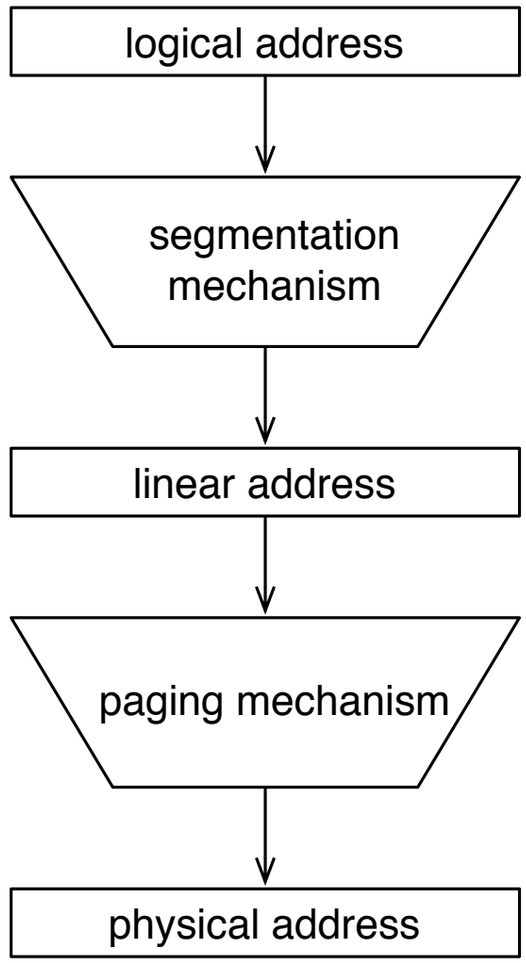
参考：x86のPTE/PDE



P	ページが物理メモリに存在しているか否か
W	書き込み可能か否か
U	一般ユーザがアクセスできるか否か
WT	キャッシュがライトスルーかライトバックか
CD	キャッシュを禁止するか否か
A	ページにアクセスすると1になる
D	ページに書き込むと1になる
AVL	OSが自由に使える（Linux, Xv6では未使用）
PPN	ページ（テーブル）の先頭アドレスの上位20ビット

参考：x86のメモリ管理機構

- CPU(MMU)はセグメンテーションとページングにより２段階の変換を行い，論理アドレスを物理アドレスに変換する.
- ただしxv6やLinuxでは，起動時などを除いてセグメンテーションは積極的に使われていない.
  - 単なる恒等写像になっている.



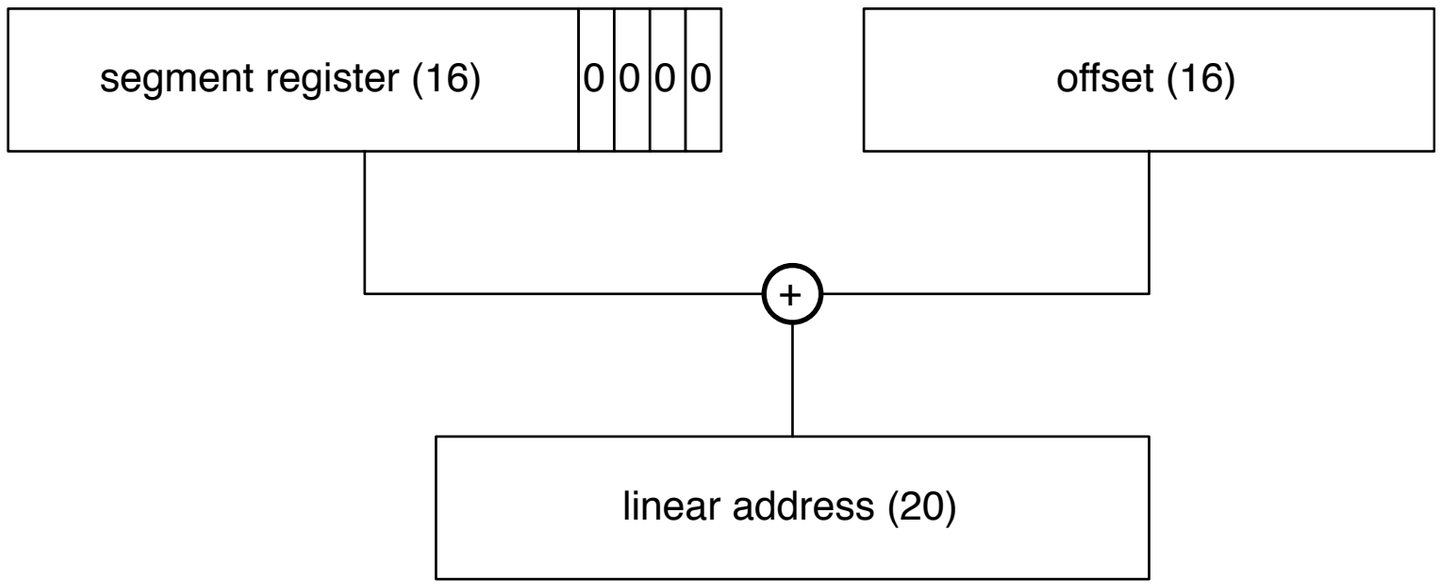
参考：x86のセグメンテーション

- 物理メモリをセグメント(segment)と呼ばれる単位に分割して管理する機構
  - セグメントの大きさは可変（固定の場合もある）
  - 各セグメントに読み書きや実行可能性等の属性を持たせることで，メモリ保護を実現する
- x86は，リアルモードとプロテクトモードで全く異なるセグメンテーションを用いる
  - リアルモードのセグメントの大きさは64KB固定で，保護機構などはない．

参考：x86のモード

- リアルモード（8086互換モード）
  - アドレス空間は1MB（20ビット）
  - CPU起動時（電源投入時）はこのモード
  - Xv6のブートローダはプロテクトモードへの移行作業を行う.
- プロテクトモード
  - アドレス空間は4GB（32ビット）
  - MMUによるメモリ管理や動作モード（特権モードやユーザモード等）が使える.
  - Xv6やLinux等のOSは通常このモードで動作する.

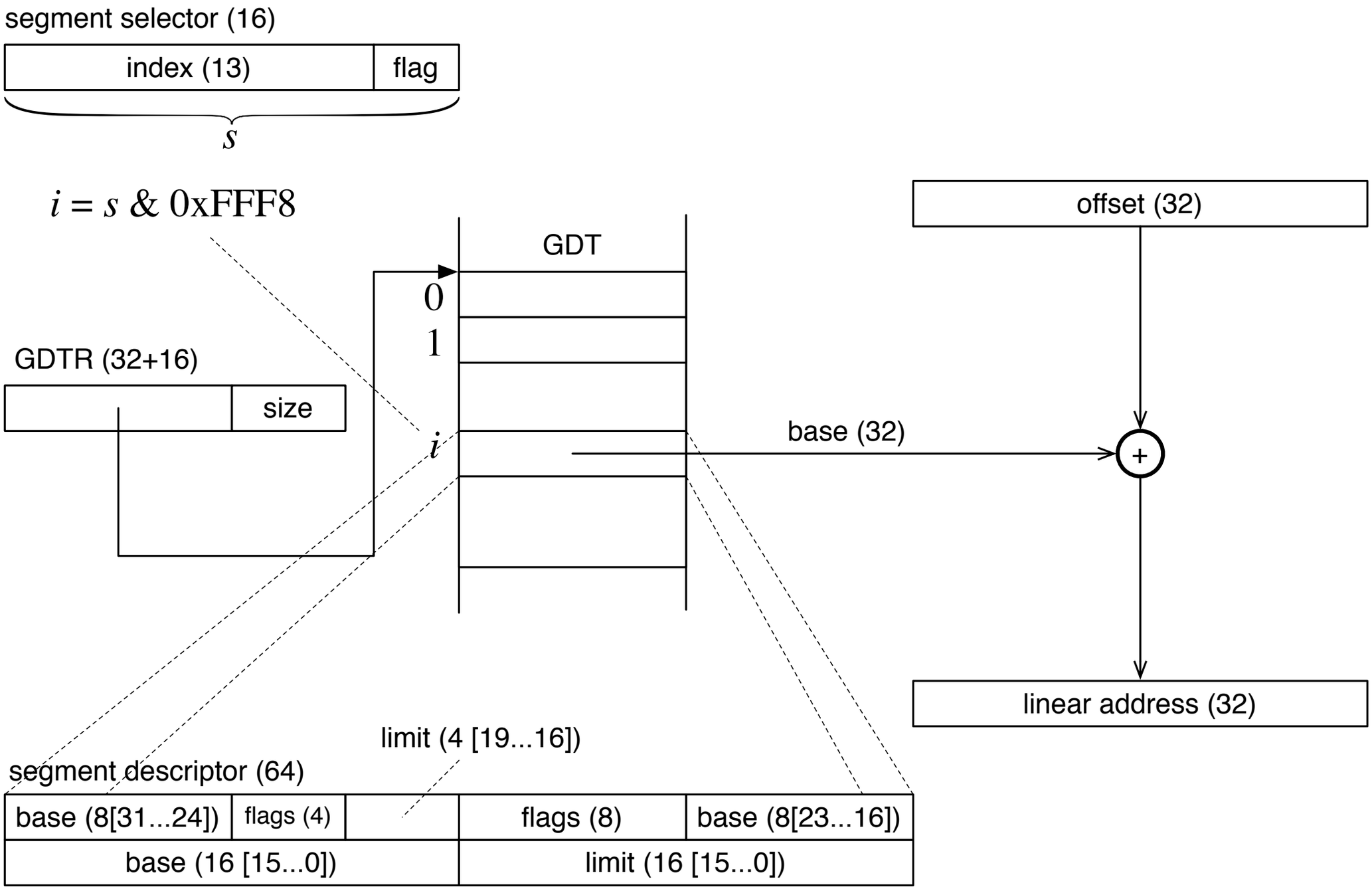
参考：x86のリアルモードにおけるセグメンテーション



- セグメントレジスタ(CS,DS,SS,etc.)の値を4ビット左シフトし，16ビットのアドレス値(offset)に足して20ビットのリニアアドレス（物理アドレス）を得る.
- セグメントの大きさは64KBに固定される.
- セグメント外へジャンプしたり，セグメント外のデータをアクセスするのは面倒（いわゆる「64KBの壁」）

参考：x86のプロテクトモードにおけるセグメンテーション

- 各セグメントの先頭アドレス（32ビット）と大きさ，属性はセグメントディスクリプタと呼ばれる8バイトの構造体で定義される.
- セグメントディスクリプタはGDT(Global Descriptor Table)と呼ばれる配列の要素である.
  - LDT (Local -)，IDT (Interrupt -)というのもある
- セグメントセレクタ(CS, DS, SS, etc.，16ビット)の下位3ビットを0にした値がGDTに対する添字であり，特定のセグメントディスクリプタを指す.
- GDTはメモリ中にあり，その先頭アドレスと大きさはGDTRレジスタで示される.

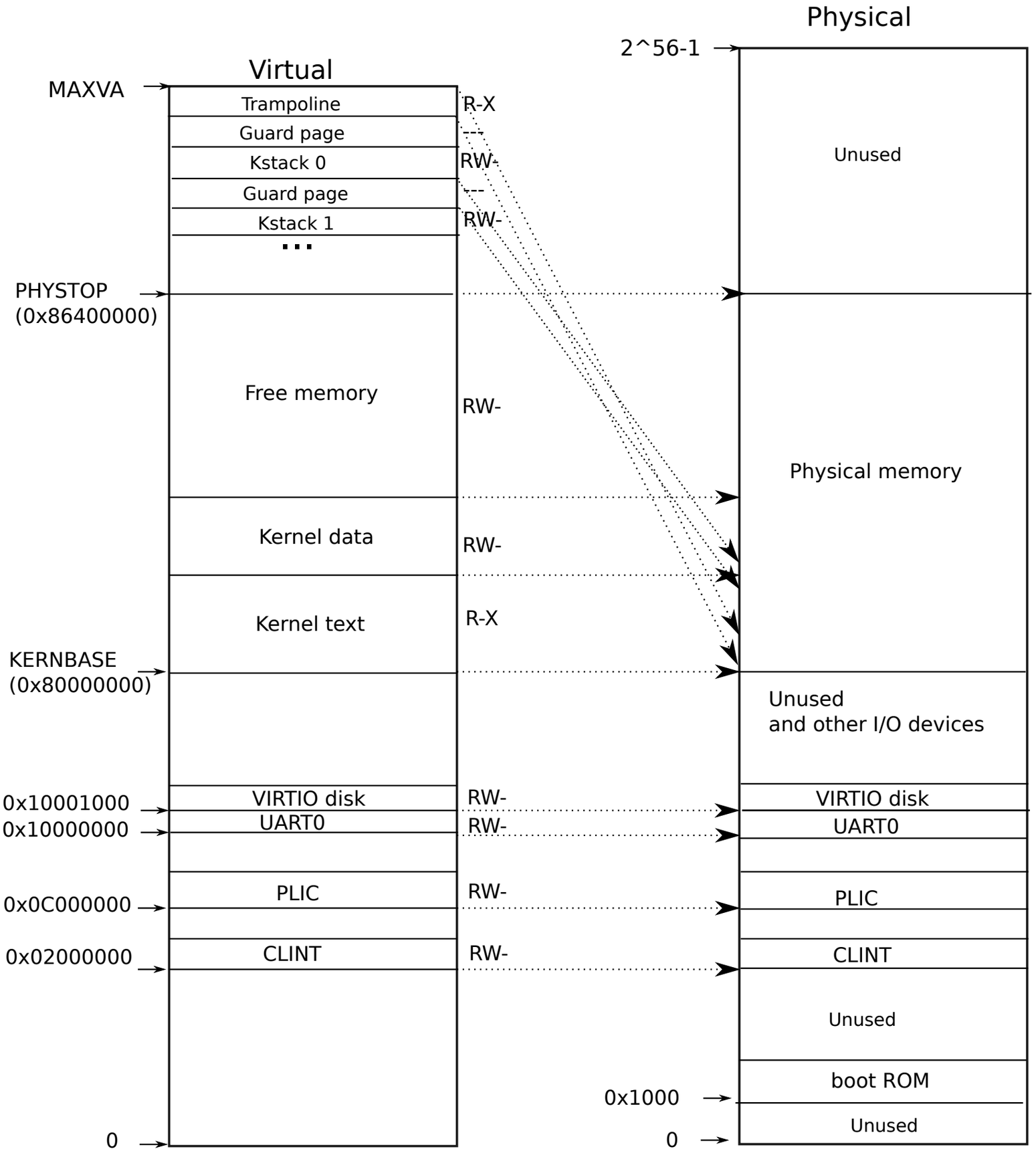

segment selector (16)
index (13)
flag
s
i = s & 0xFFF8
GDT
0
1
i
GDTR (32+16)
size
offset (32)
base (32)
+
linear address (32)
limit (4 [19...16])
segment descriptor (64)
base (8[31...24])
flags (4)
flags (8)
base (8[23...16])
base (16 [15...0])
limit (16 [15...0])

参考：x86の
セグメントディスクリプタ

- 以下の要素を持つ64ビット構造体
  - base (32): セグメントの開始アドレス
  - limit (20): セグメントのサイズ
    - 0～limit値までがセグメントとして使える領域となる.
    - 1B単位か4KB単位かをGフラグで選択できる.
  - flags: 各種属性
    - type (4): セグメントの種類
      - 読み出し，書き込み，実行が可能か否か等
    - dpl (2): 特権レベル
- メモリ中で定義されるが，実際にはCPU内に読み込まれて使用される.

xv6：物理メモリ空間

- 0～KERNBASE-1(=0x7FFFFFFF)
  - I/Oや割り込みコントローラ等
    - メモリマップドI/O
- KERNBASE～PHYSTOP
  - KERNBASE (0x80000000)
    - カーネルのコードの起点
  - PHYSTOP (0x86400000)
    - 物理メモリ(RAM 128MB)の上限
    - PHYSTOP = KERNBASE + 128 * 1024 * 1024


Physical
2^56-1
Virtual
MAXVA
Trampoline
R-X
Guard page

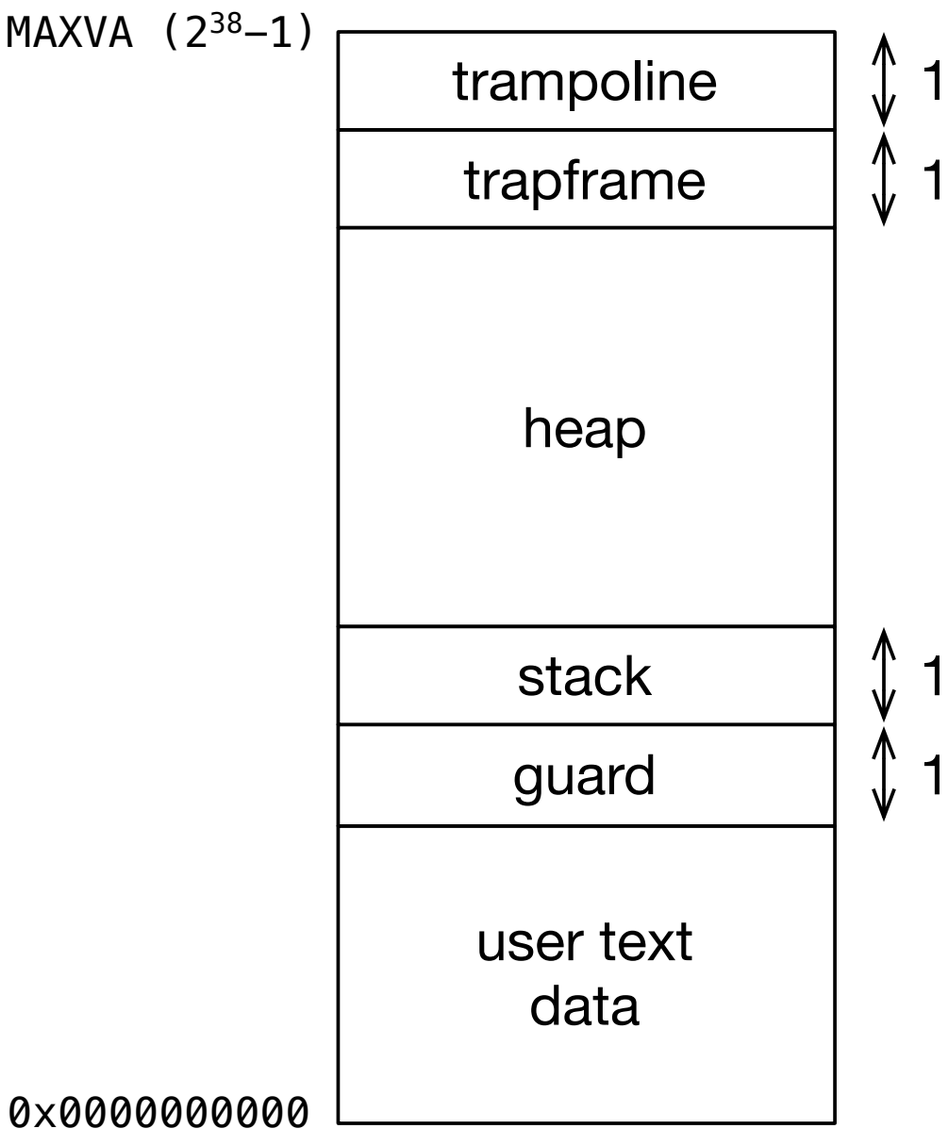
Kstack 0
RW-
Guard page

Kstack 1
RW-
...
Unused
PHYSTOP
(0x86400000)
Free memory
RW-
Physical memory
Kernel data
RW-
Kernel text
R-X
KERNBASE
(0x80000000)
Unused
and other I/O devices
0x10001000
VIRTIO disk
RW-
VIRTIO disk
0x10000000
UART0
RW-
UART0
0x0C000000
PLIC
RW-
PLIC
0x02000000
CLINT
RW-
CLINT
Unused
boot ROM
0x1000
Unused
0

プロセスのメモリ空間

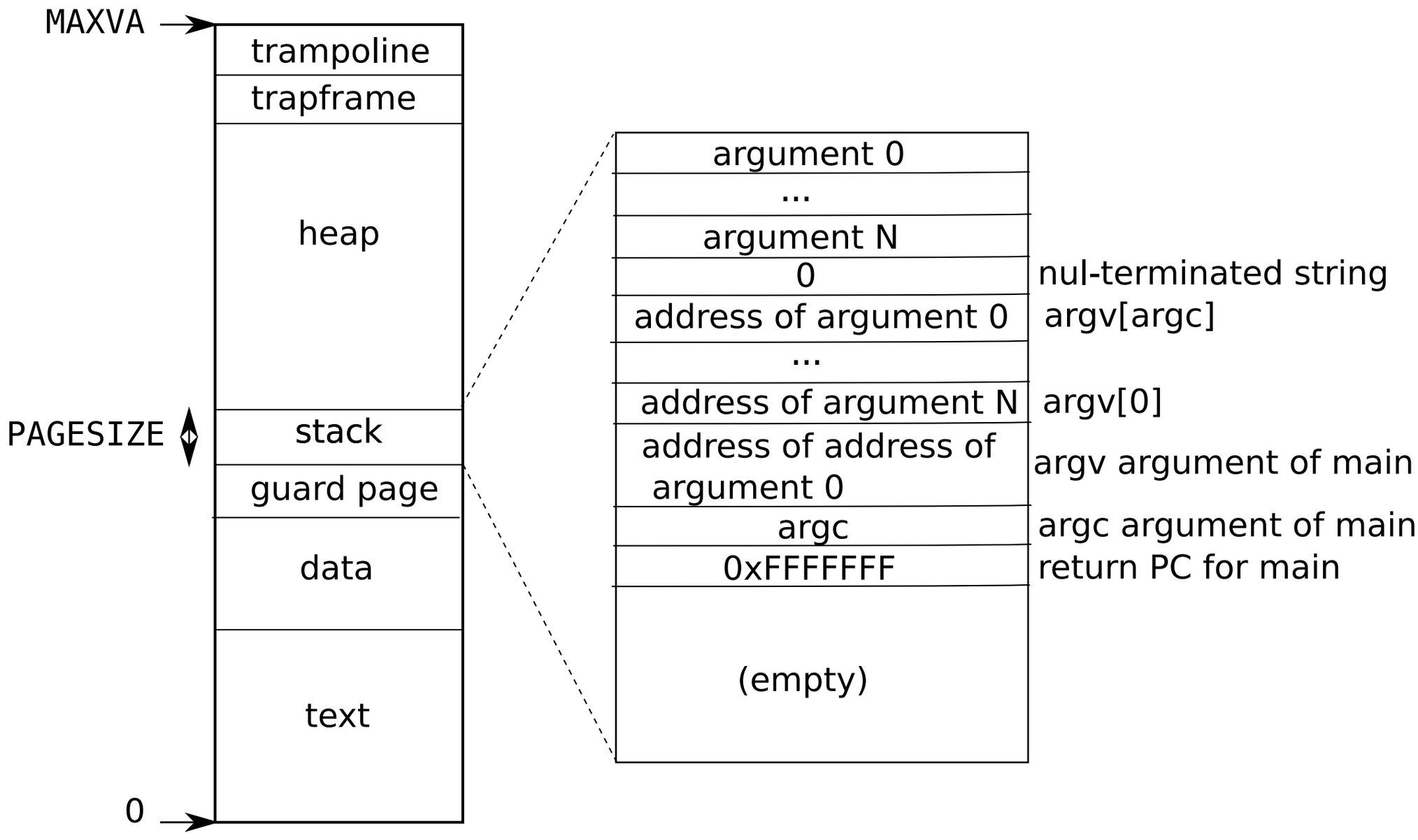
- 論理アドレス：0〜MAXVA-1
  - MAXVA = 2^{38} = 0x4000000000
    - Sv39を用いているがビット38を未使用としている．したがって論理アドレス空間の大きさは256GB.
      - Sv39では，論理アドレスの上位25ビット（ビット39〜63）はビット38のコピー（符号拡張）である必要がある．最上位1ビットを未使用とする理由はその処理が面倒なため.
- メモリマップ
  - 最下位から：テキスト領域，データ領域，ガードページ，スタック，ヒープ
  - 最上位から：トランポリン，トラップフレーム

プロセスのメモリ空間



- アドレス：64ビット
  - 有効なのは38ビット
- user text & data
  - プログラム
  - 定数などコンパイル時に大きさが決まっているデータ
- stack
  - コールスタック
- heap
  - 実行時に獲得されるメモリ
- trampoline
  - カーネル空間との切り替えに使うコード
  - カーネル空間と共有
- trapframe
  - システムコールの引数等

プロセスのメモリレイアウト



カーネル内のメモリ割り当て

- 関数kallocで1ページを確保する(kalloc.c)
  - p37の"(Free Memory)"とある領域から割り当てる
- 割り当てたメモリの用途
  - カーネルが利用するメモリ
    - カーネルスタック，パイプバッファ
  - ユーザプロセスが利用するメモリ
    - ページ，ページテーブル
- 空きページはリストとして管理
  - いわゆる素朴なメモリ管理の一種

プロセス毎のメモリ割り当て(1)

- allocproc (proc.c)
  - プロセスを作成する関数
  - カーネルスタックの割り当て
    - プロセス構造体のkstackにkallocで1ページ(4096バイト)を割り当てる
    - 割り込み時に作成されるトラップフレームもこの中に作られる

プロセス毎のメモリ割り当て(2)

- ページディレクトリ (proc->pgdir)
  - プロセスのメモリ空間を定義する
    - カーネルコードとデータのエントリ
    - ユーザメモリ空間のエントリ
  - 割り当てのタイミング
    - forkの中でcopyuvmによって親プロセスのメモリ空間のコピーとして作成される
    - execの中でallocuvmによって新規に割り当てられる（その際古いメモリ空間は消去される）

まとめ

- メモリ管理
  - 素朴なメモリ管理技法
  - xv6のメモリ管理
  - 仮想記憶
    - スワップ領域，ページイン／アウト，犠牲ページとページ置換アルゴリズム，LRU，参照ビットと汚れビット
    - 応用
      - メモリマップドファイル
      - Copy on Write
      - 共有メモリ