Operating System Principles

操作系统原理

Memory Management

李旭东

leexudong@nankai.edu.cn

Nankai University

Objectives

- No Memory Abstraction
- Basic Memory Management
- ~~*Virtual Memory Management*~~

Parkinson's law

Programs expand to fill the memory available to hold them!

Memory Management

- Memory is an important resource that must be carefully managed

While the average home computer nowadays has a thousand times as much memory as the IBM 7094, the largest computer in the world in the early 1960's

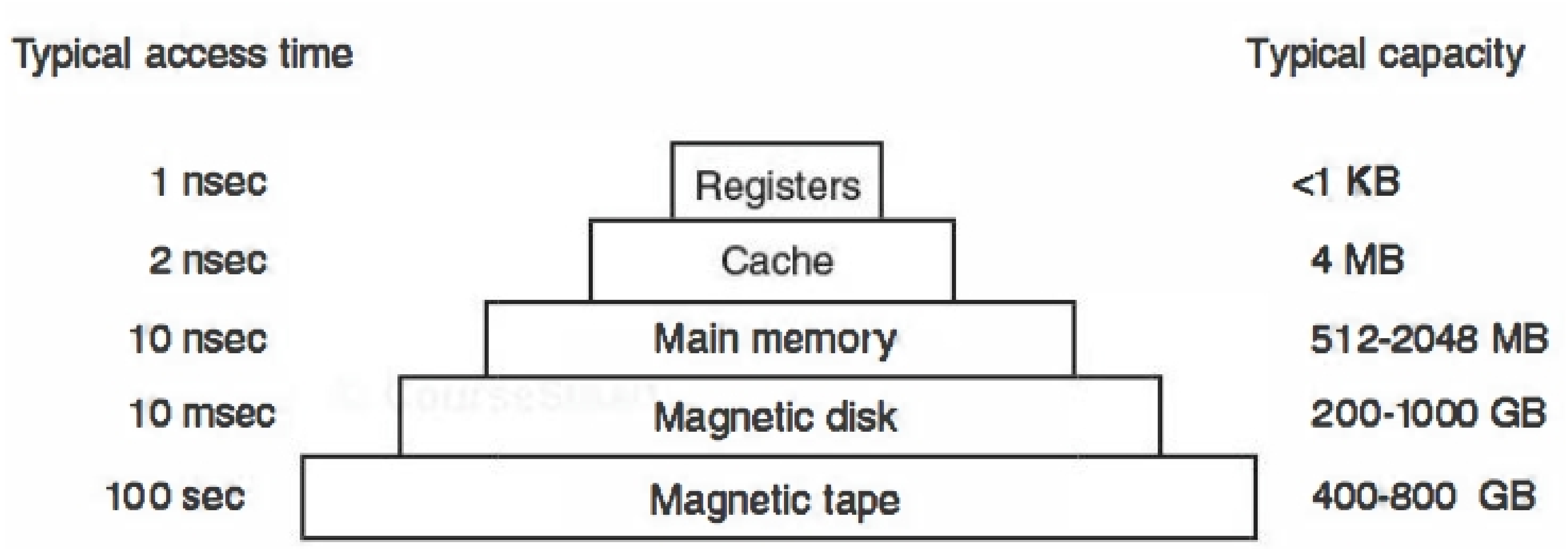
- Memory hierarchy
  - *Volatile* cache memory

    a small amount,very fast,expensive
  - Volatile main memory(RAM)

    tens of megabytes, medium-speed, medium-price
  - Nonvolatile disk storage

    tens or hundreds of gigabytes, slow, cheap

Memory Hierarchy



Memory Management

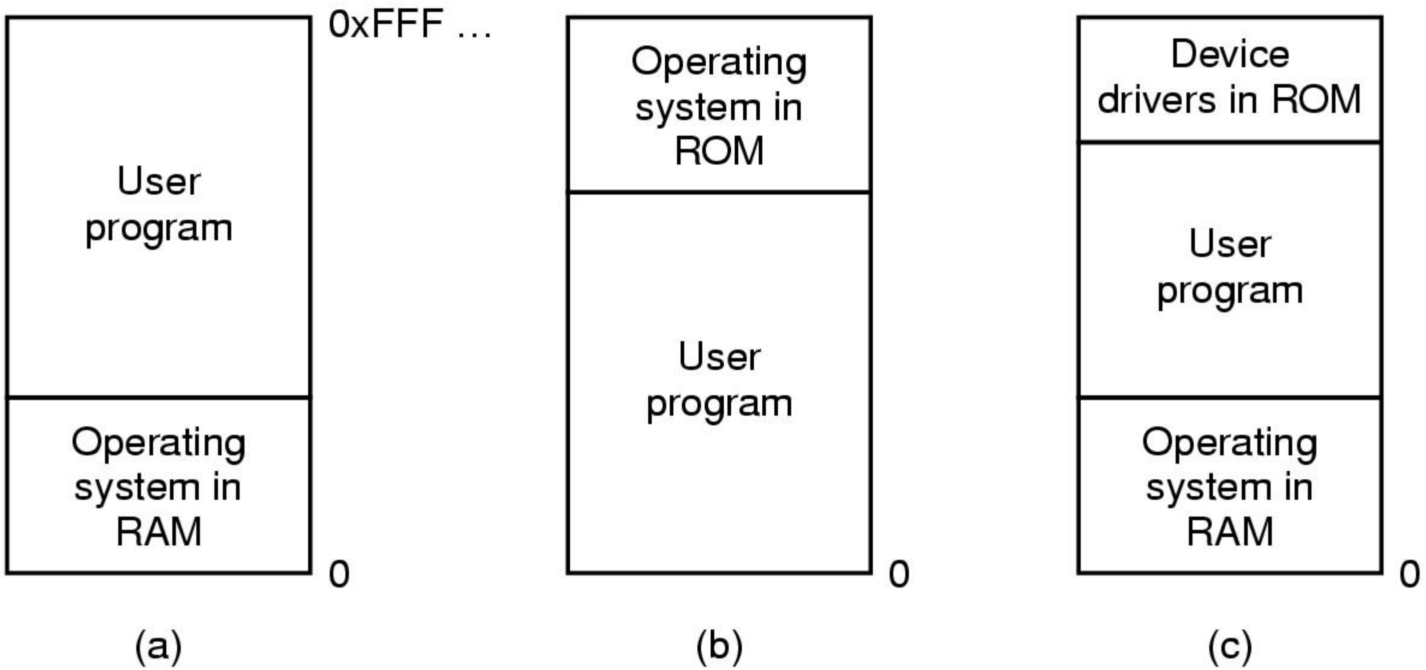
- 1. extend main memory
- 2. control data transmission between main memory and storage
- 3. main memory allocation and revoke
- 4. main memory share and protection

No Memory Abstraction

No Memory Abstraction

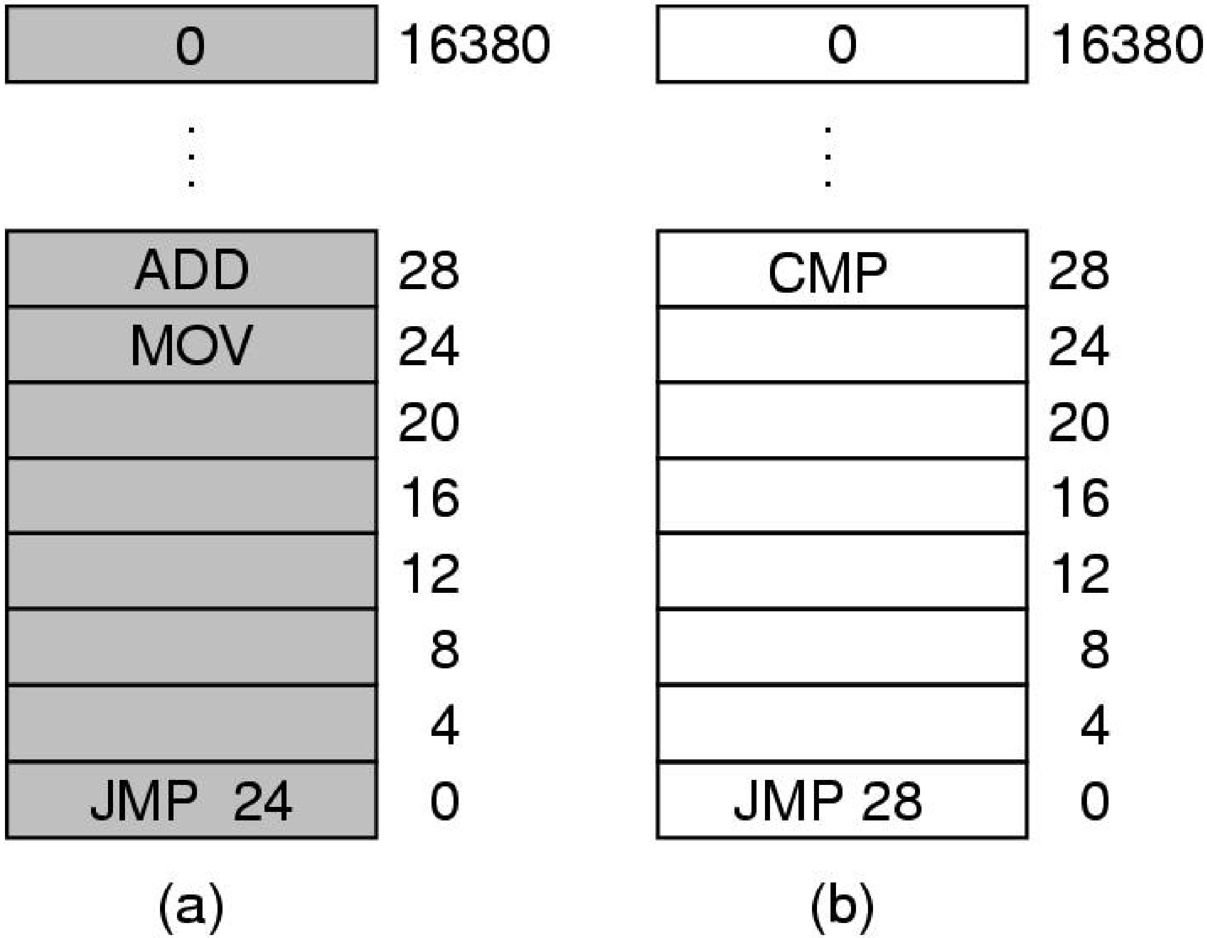
- early mainframe computers (<1960)
- early minicomputers (<1970)
- early personal computers (<1980)

RAM(Random Access Memory), ROM(Read-Only Memory)



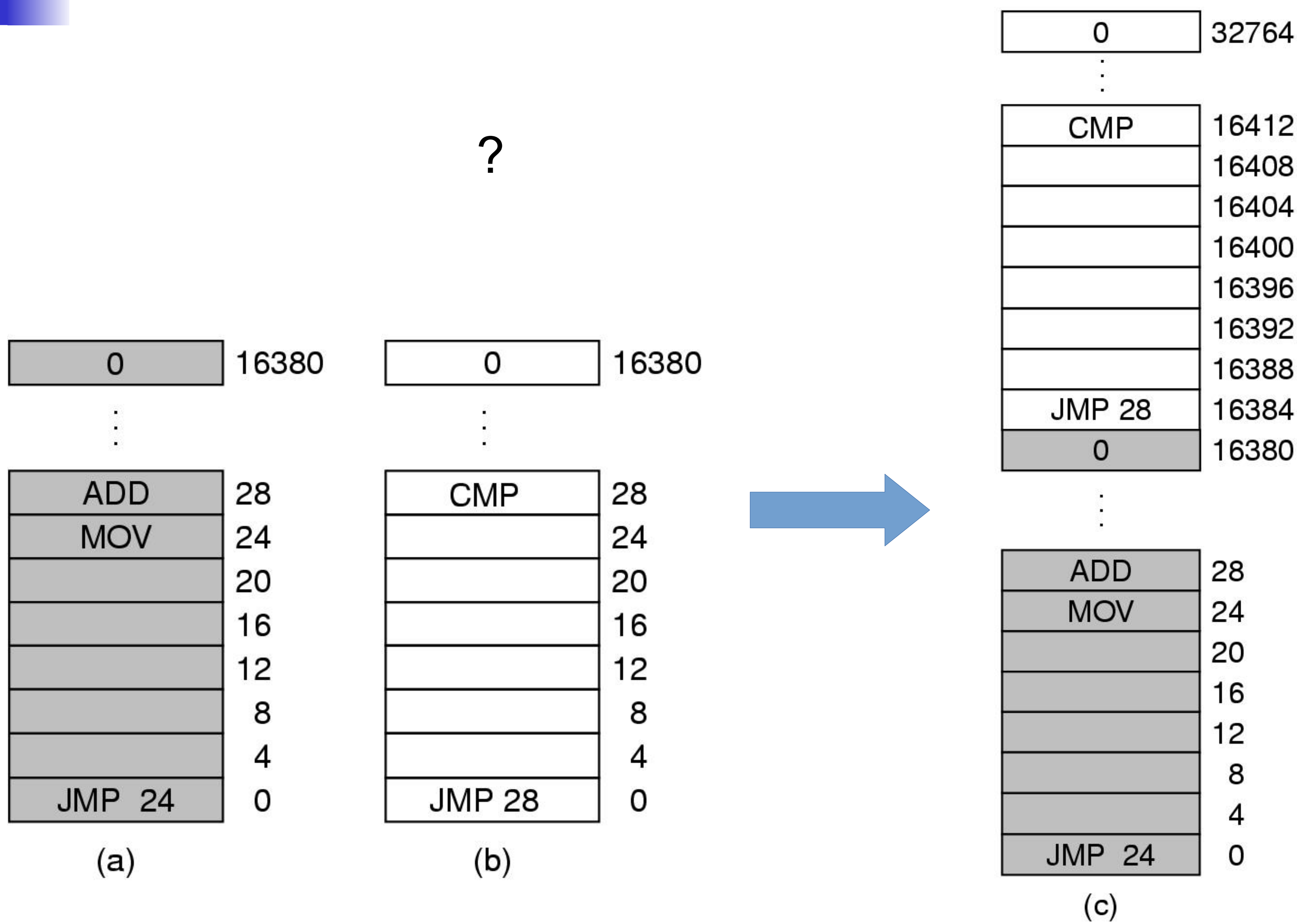
Three simple ways of organizing memory with an operating system and one user process.

Issues of No Memory Abstraction



Two Processes

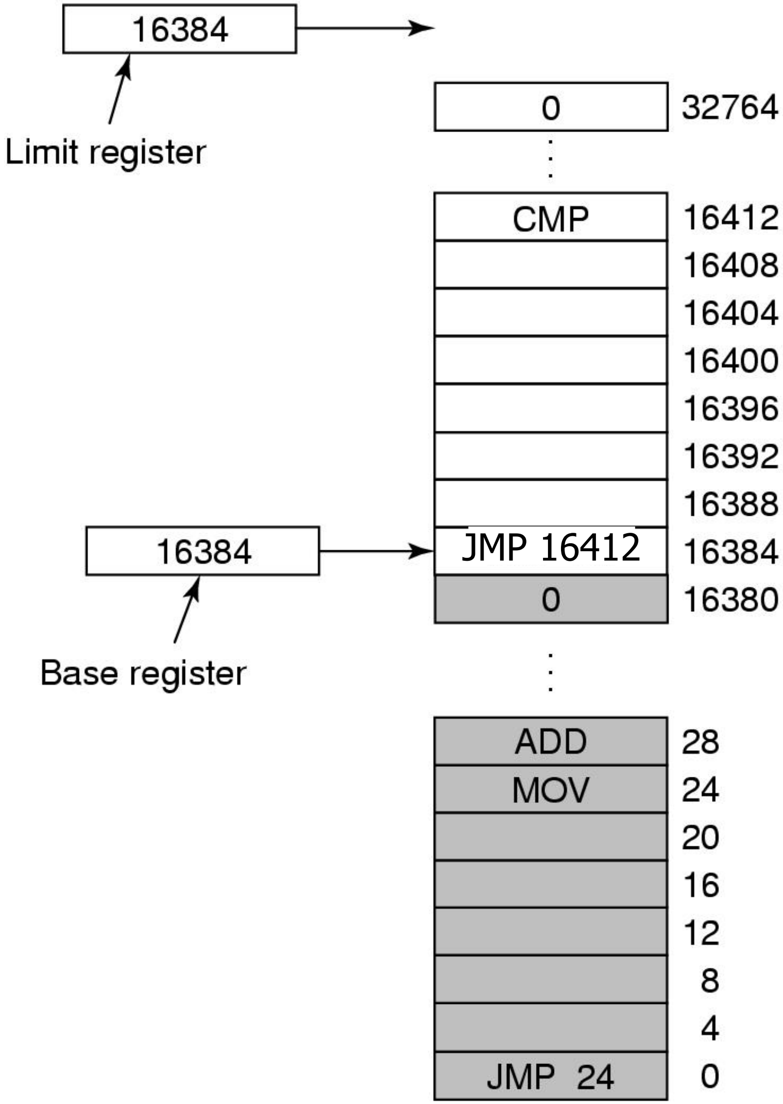
Issues of No Memory Abstraction



Issues of No Memory Abstraction

1.separate address space

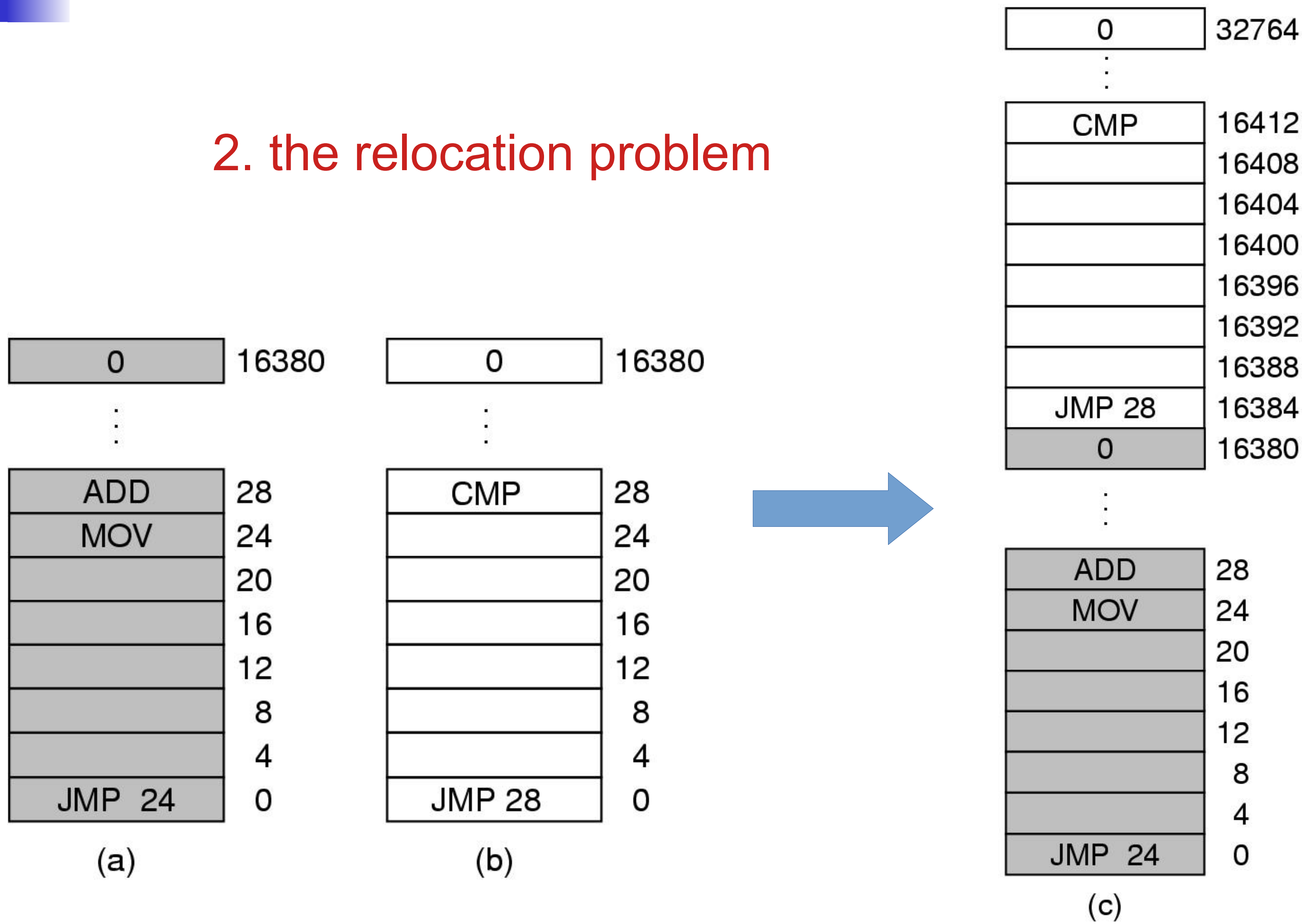
Base register
Limit register



(c)

Issues of No Memory Abstraction

2. the relocation problem



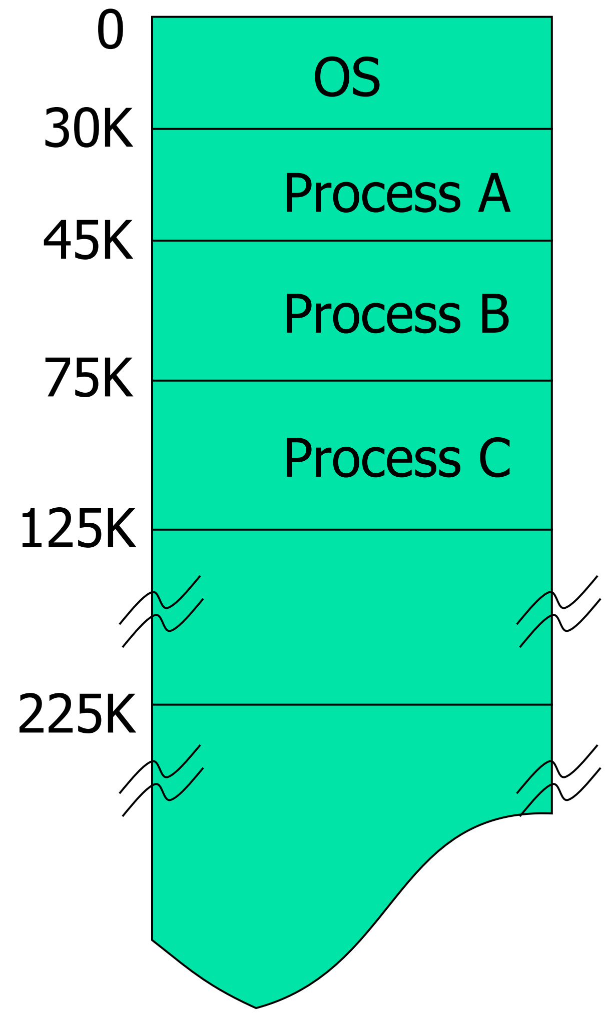
Basic Memory Management
- contiguous 连续 allocation

Multi-programming with Fixed Partitions

- Partition
  - Fixed, size

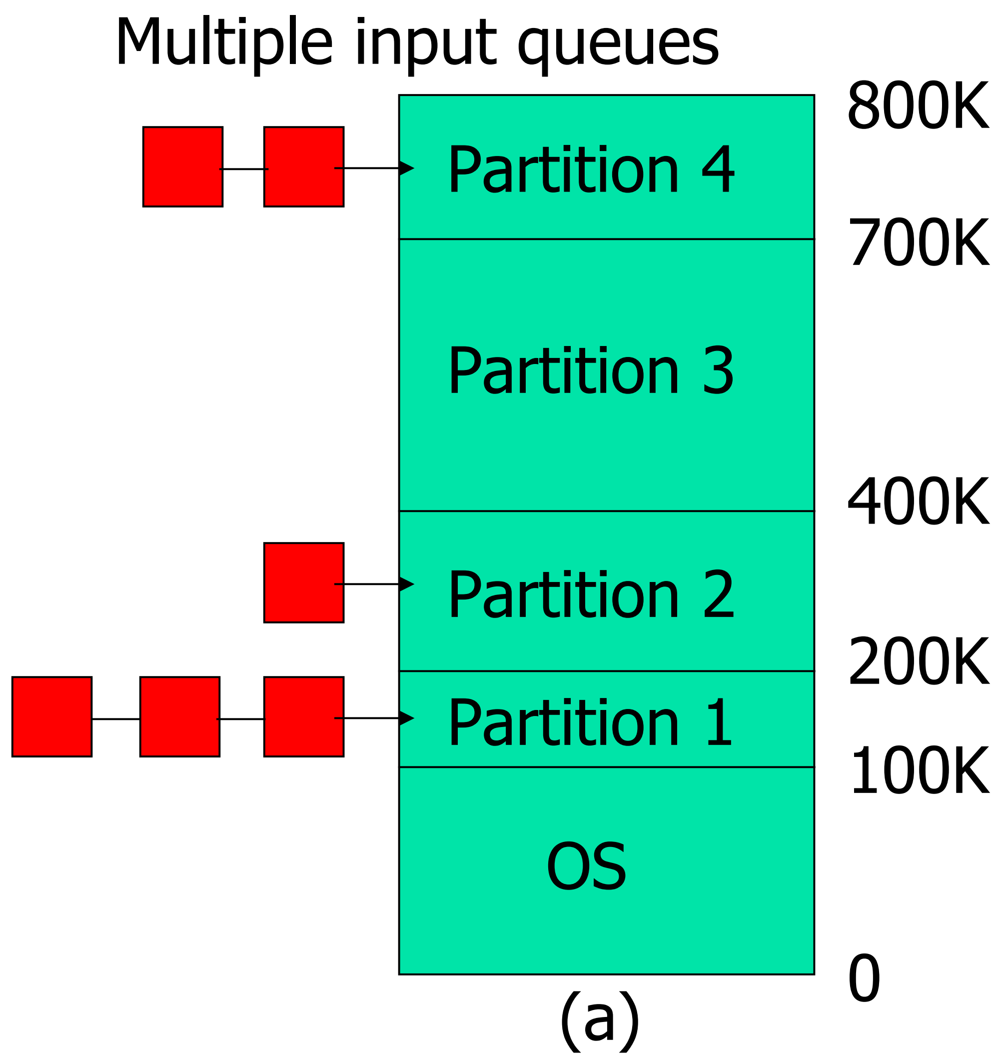
ID	Size (KB)	Start Addr (K)	State
1	15	30	used
2	30	45	used
3	50	75	used
4	100	125	available

(a) partition desc table



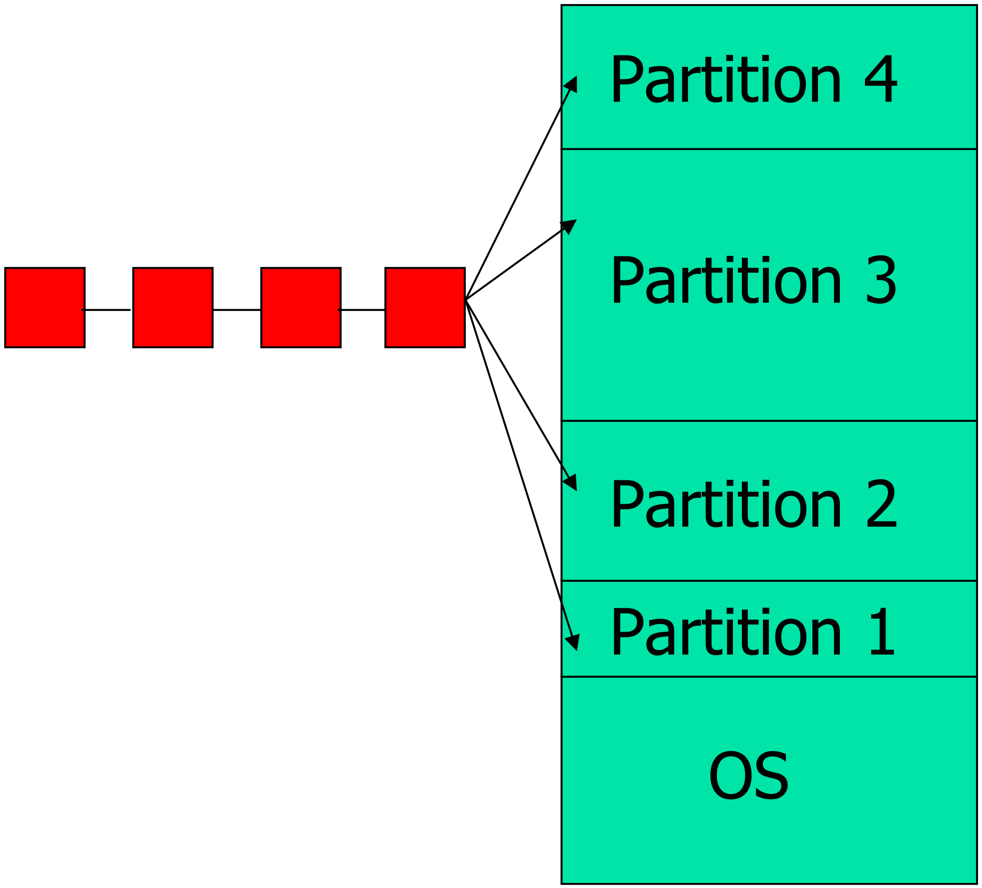
(b) memory layout

Multi-programming with Fixed Partitions



Multi-programming with Fixed Partitions

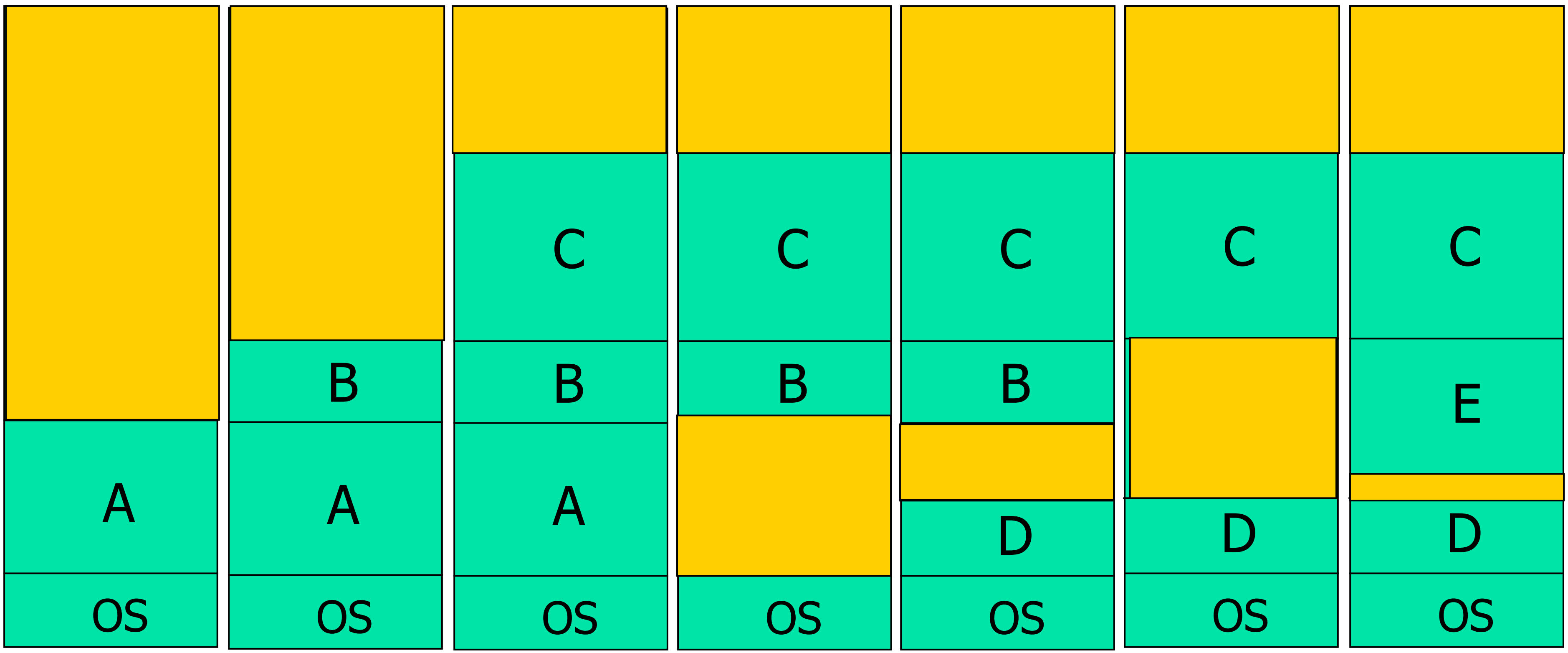
Single input queque



(b)

Multi-programming with Variable Partitions

- Allocate a contiguous partition dynamically



(a) (b) (c) (d) (e) (f) (g)

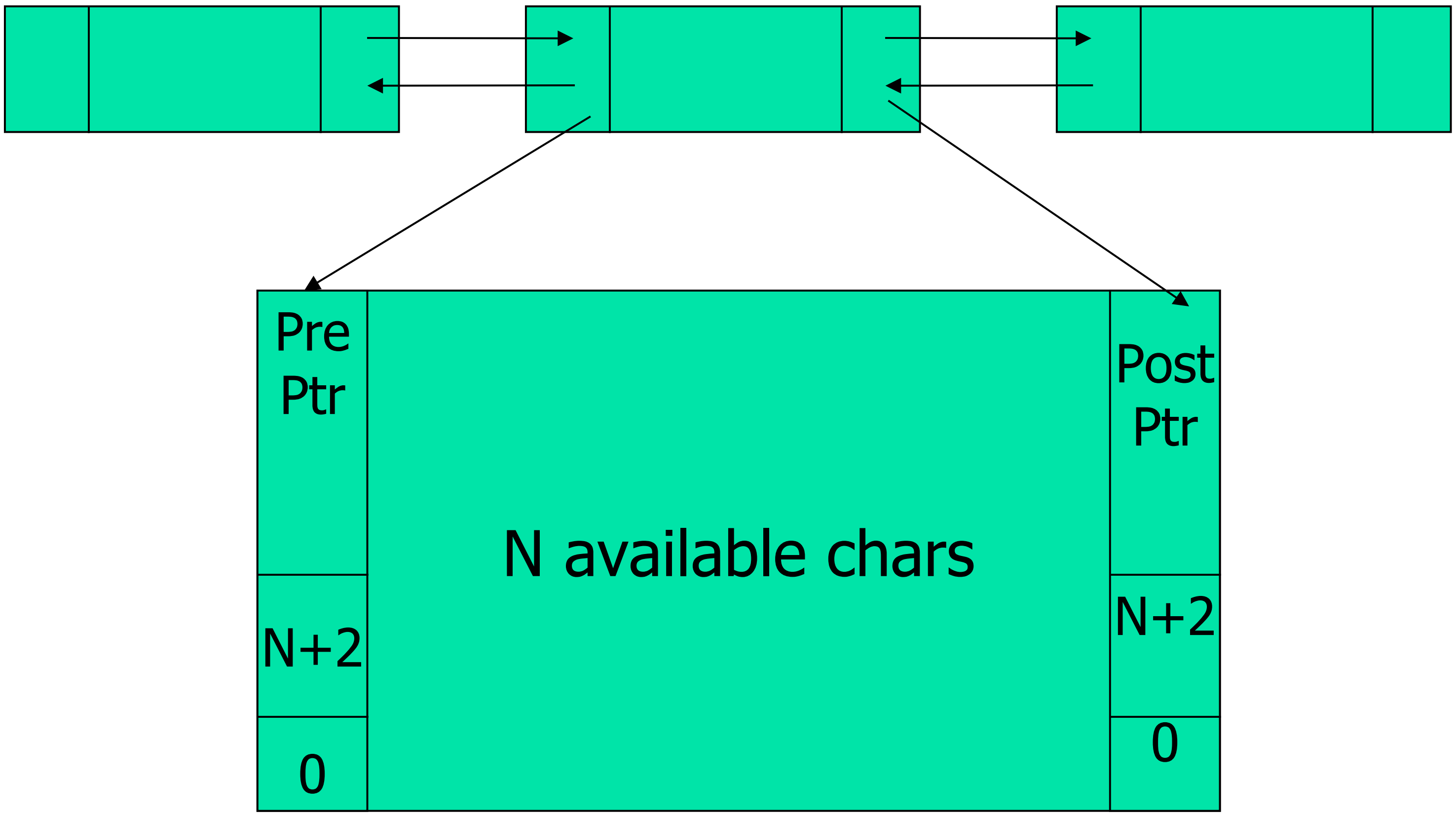
Multi-programming with Variable Partitions

- Structure
- Algorithm
- Allocating and Revoking Procedures

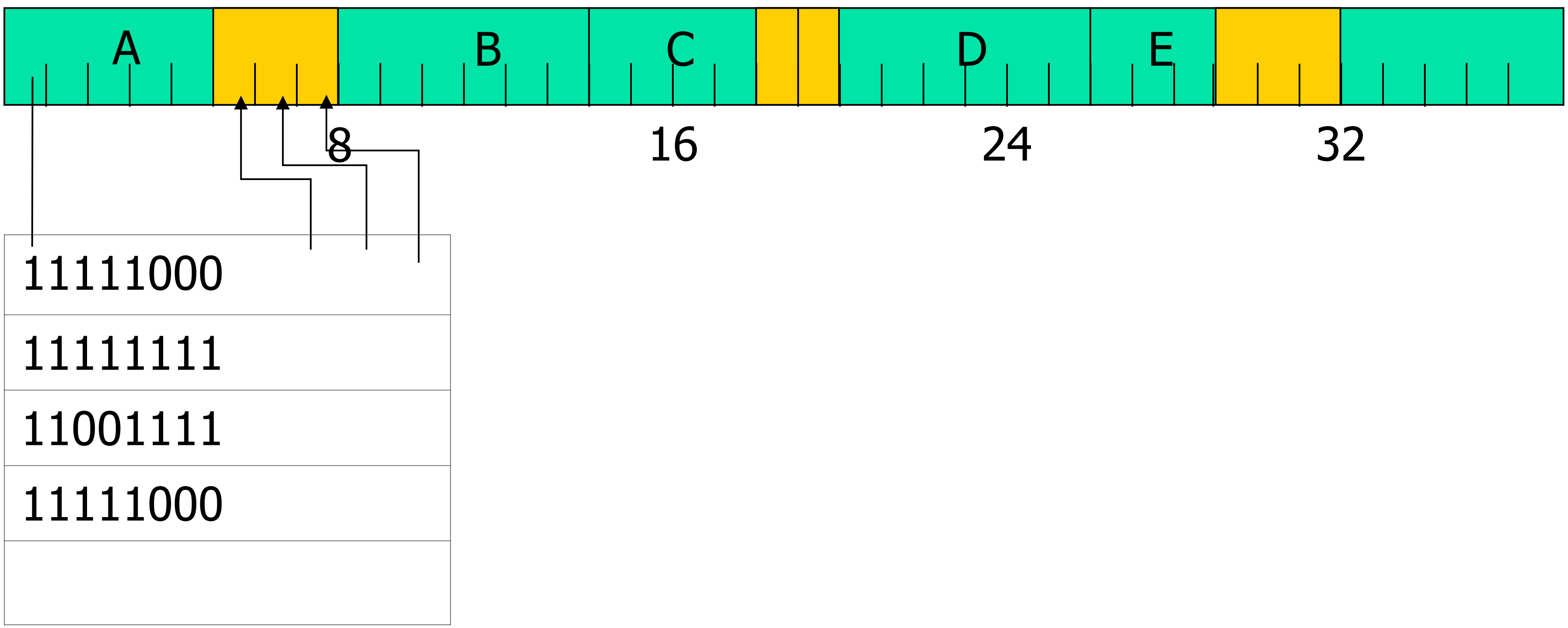
Variable Partitions: Partition Array Table

ID	Size (KB)	Start Addr (K)	state
1	64	44	available
2	24	132	available
3	40	210	used
4	30	270	available
5	…	…	…

Variable Partitions: Inline Linked Structure

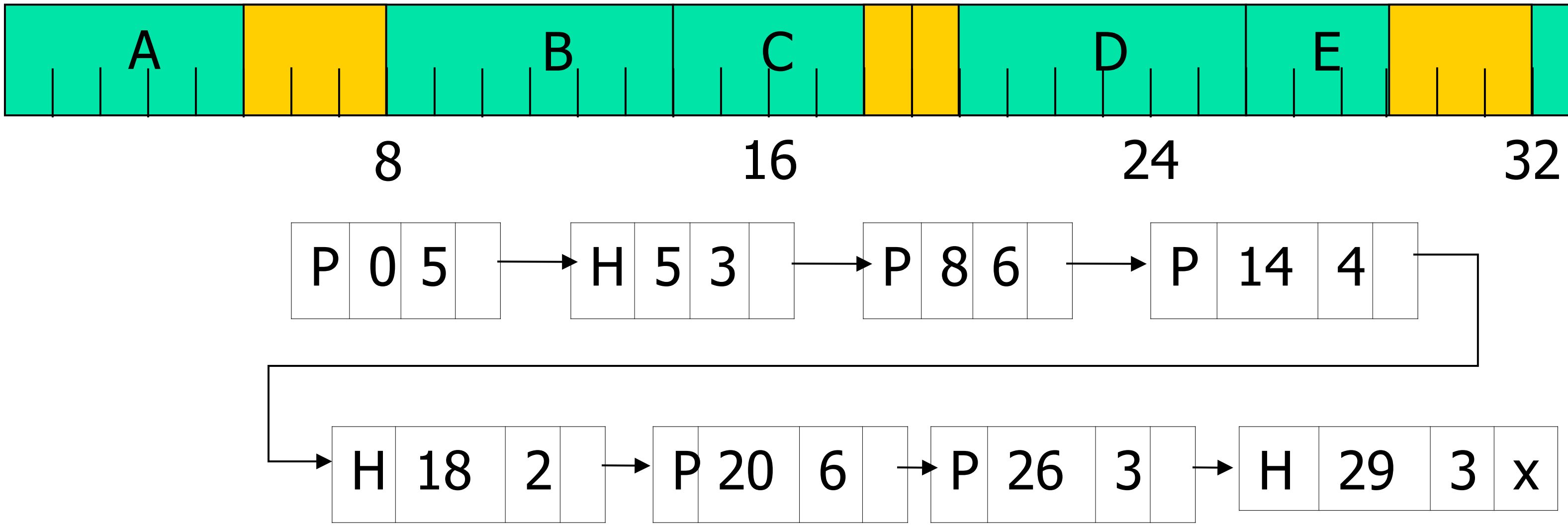


Variable Partitions: Bitmap 位图



Variable Partitions: Linked List

P: Process
H: Free space



Algorithms of Variable Partitions

- First Fit: FF
- Next FF
- Best Fit: BF
  - 最佳适应算法
  - external fragmentation
    - 碎片
- Worst Fit: WF

Algorithms of Variable Partitions

- Quick Fit
  - Multi-queues for 4KB, 8KB,16KB free contiguous space
  - Advantages
  - Disadvantage
    - Overhead of merging free partitions

Algorithms of Variable Partitions

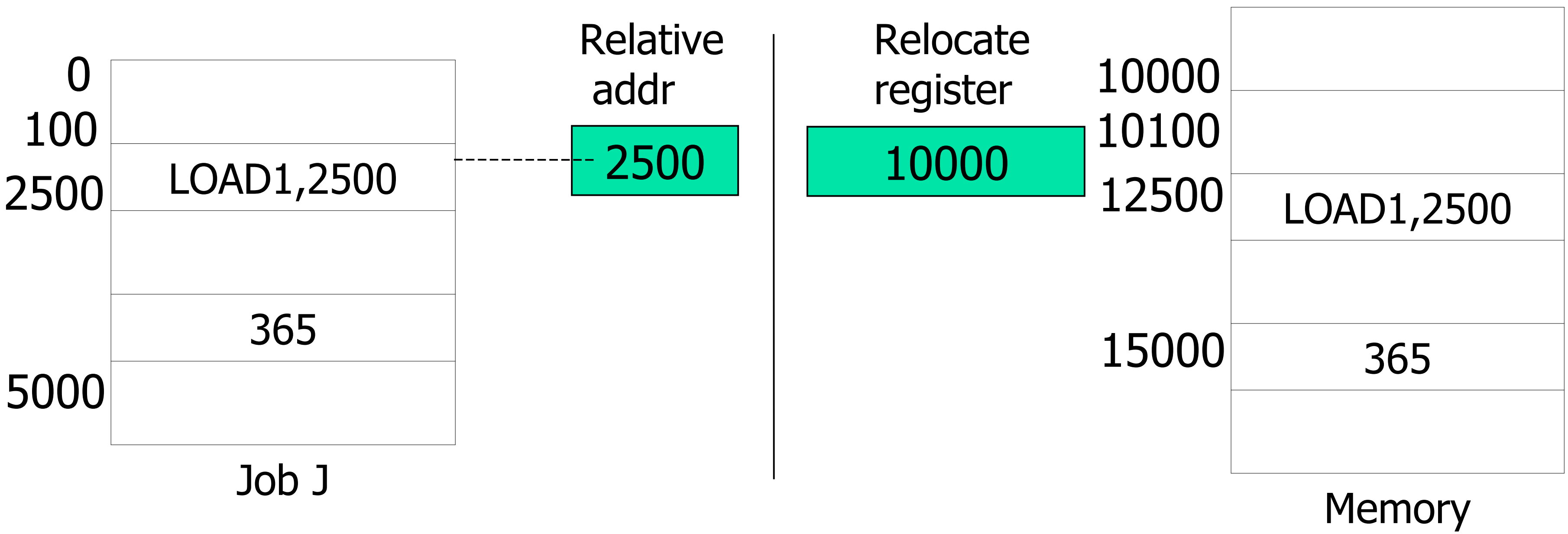
- Buddy memory allocation
  - 伙伴式的内存管理
  - 1963,Harry Markowitz,who won the 1990 Nobel Memorial Prize in Economics
  - each block is subdivided into two smaller blocks
  - 2^i
  - internal fragmentation

Case: buddy memory allocation

$2^4 \rightarrow 2^3 \rightarrow 2^2 \rightarrow 2^1 \rightarrow 2^0 \rightarrow A \rightarrow C \rightarrow B \rightarrow$

<table>
<tr><th>Step</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th><th>64K</th></tr>
<tr><td>1</td><td colspan="16">2^4</td></tr>
<tr><td>2.1</td><td colspan="8">2^3</td><td colspan="8">2^3</td></tr>
<tr><td>2.2</td><td colspan="4">2^2</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>2.3</td><td colspan="2">2^1</td><td colspan="2">2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>2.4</td><td>2^0</td><td>2^0</td><td colspan="2">2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>2.5</td><td>A: 2^0</td><td>2^0</td><td colspan="2">2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>3</td><td>A: 2^0</td><td>2^0</td><td colspan="2">B: 2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>4</td><td>A: 2^0</td><td>C: 2^0</td><td colspan="2">B: 2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>5.1</td><td>A: 2^0</td><td>C: 2^0</td><td colspan="2">B: 2^1</td><td colspan="2">2^1</td><td colspan="2">2^1</td><td colspan="8">2^3</td></tr>
<tr><td>5.2</td><td>A: 2^0</td><td>C: 2^0</td><td colspan="2">B: 2^1</td><td colspan="2">D: 2^1</td><td colspan="2">2^1</td><td colspan="8">2^3</td></tr>
<tr><td>6</td><td>A: 2^0</td><td>C: 2^0</td><td colspan="2">2^1</td><td colspan="2">D: 2^1</td><td colspan="2">2^1</td><td colspan="8">2^3</td></tr>
<tr><td>7.1</td><td>A: 2^0</td><td>C: 2^0</td><td colspan="2">2^1</td><td colspan="2">2^1</td><td colspan="2">2^1</td><td colspan="8">2^3</td></tr>
<tr><td>7.2</td><td>A: 2^0</td><td>C: 2^0</td><td colspan="2">2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>8</td><td>2^0</td><td>C: 2^0</td><td colspan="2">2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>9.1</td><td>2^0</td><td>2^0</td><td colspan="2">2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>9.2</td><td colspan="2">2^1</td><td colspan="2">2^1</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>9.3</td><td colspan="4">2^2</td><td colspan="4">2^2</td><td colspan="8">2^3</td></tr>
<tr><td>9.4</td><td colspan="8">2^3</td><td colspan="8">2^3</td></tr>
<tr><td>9.5</td><td colspan="16">2^4</td></tr>
</table>

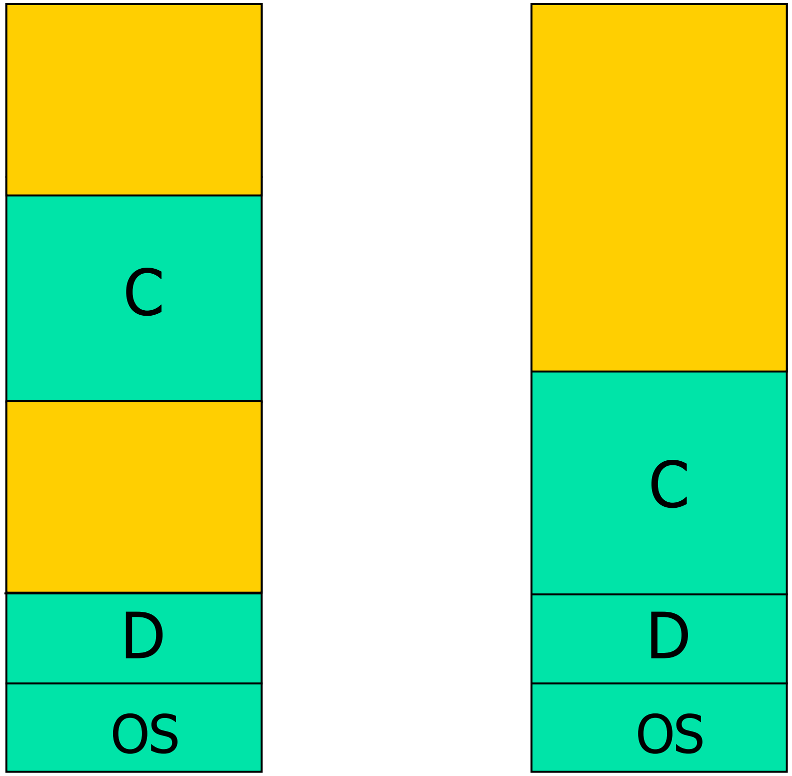
Dynamical Relocation



Memory Compaction

- 紧凑

?fragmentation

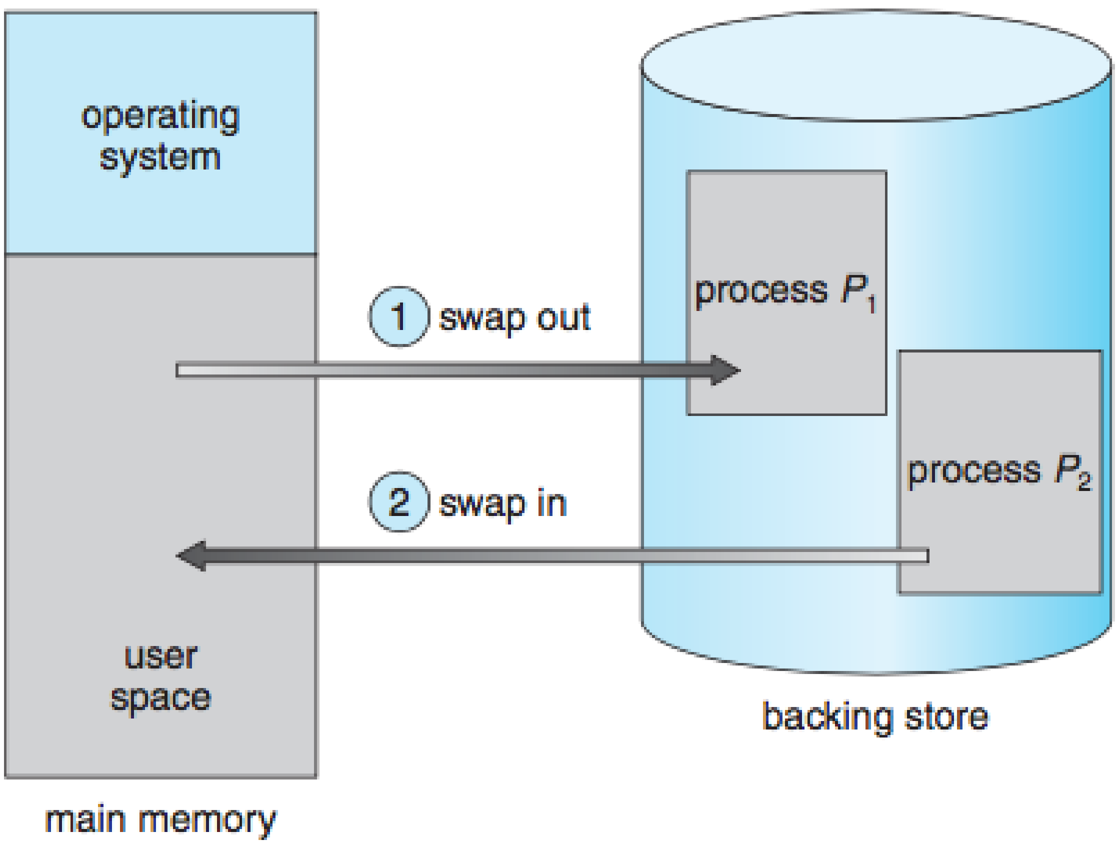


(f)
Before comp.

(f*)
After comp.

Swapping 交换

- bring in each process in its entirety, running it for a while, then putting it back on the disk

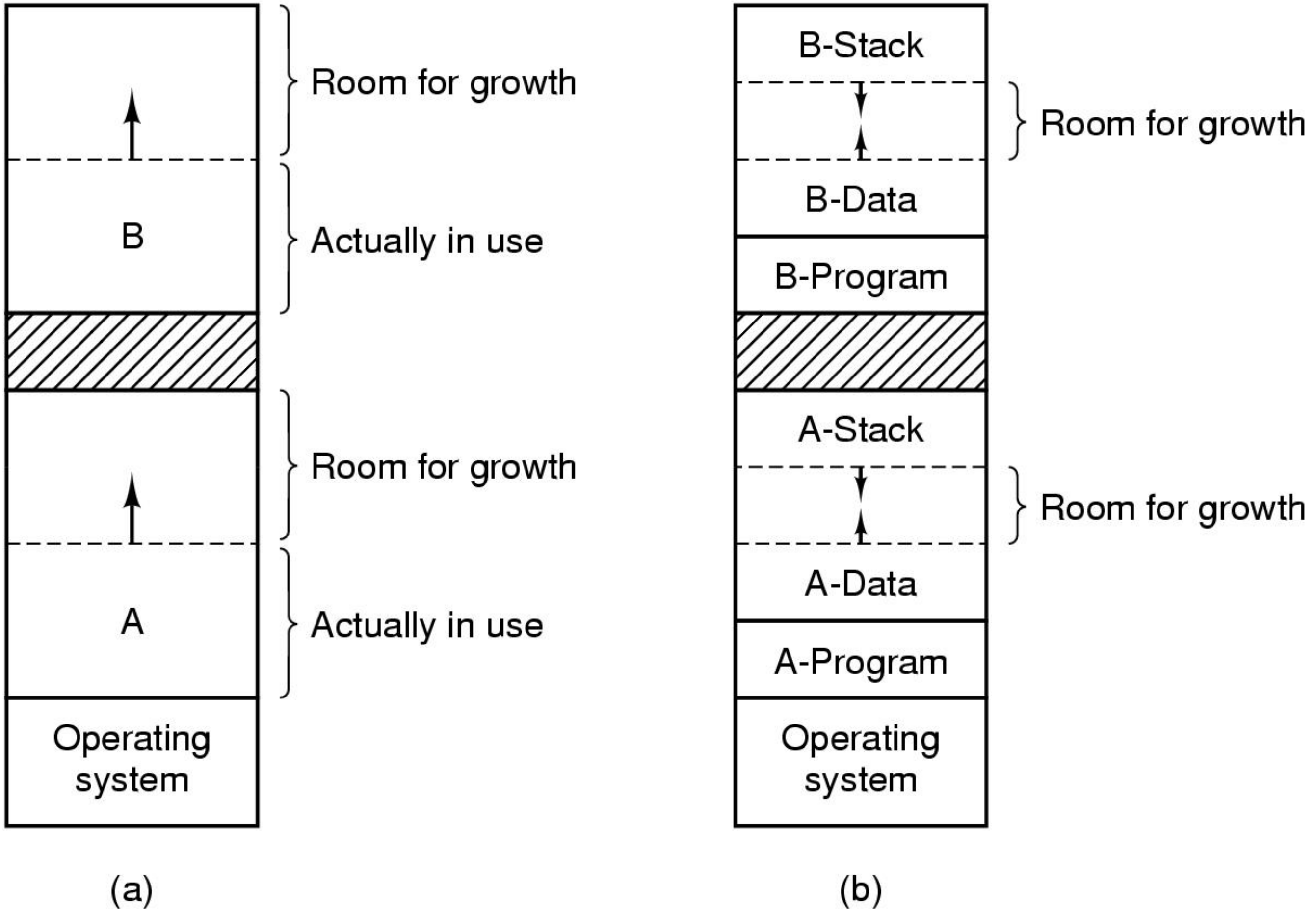


Overlay 覆盖

- Replacement of a block of stored instructions or data with another
- a way to describe sections which are to be loaded as part of a single memory image but are to be run at the same memory address

```
OVERLAY [start] : [NOCROSSREFS] [AT ( ldaddr )]
  {
    secname1
      {
        output-section-command
        output-section-command
        ...
      } [:phdr...] [=fill]
    secname2
      {
        output-section-command
        output-section-command
        ...
      } [:phdr...] [=fill]
    ...
  } [>region] [:phdr...] [=fill] [,]
```

Issues of Variable Partitions



(a) Allocating space for growing data segment.

(b) Allocating space for growing stack, growing data segment.

Basic Memory Management - discrete allocation

Discrete Memory Allocation

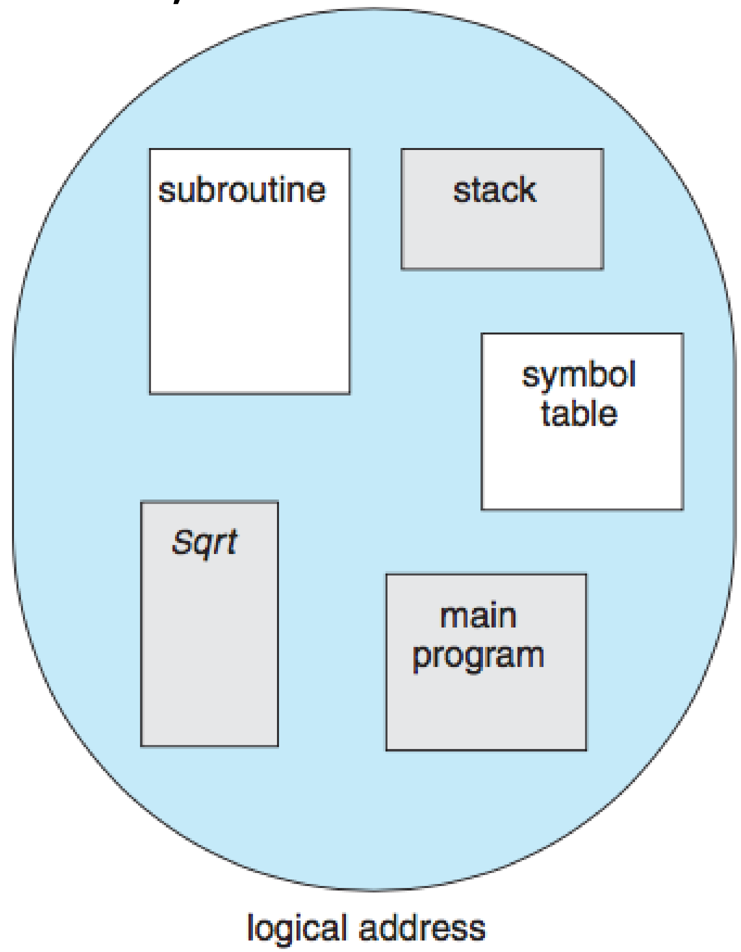
- Segmentation
  - 分段
- Paging
  - 分页

Segmentation 分段

two tuple:<segment-number, offset>

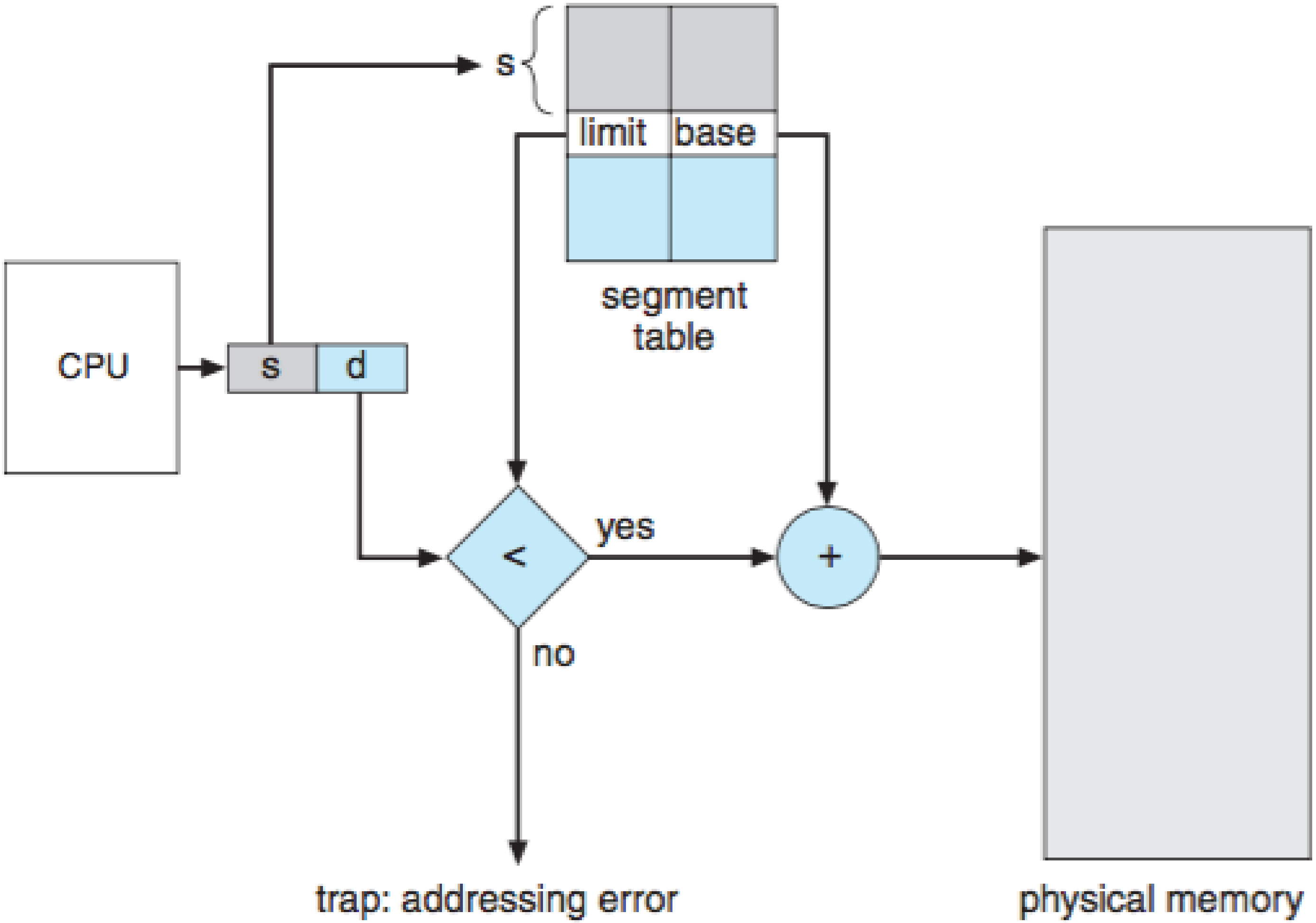
- Logical Segmentation
  - 1. The code
  - 2. Global variables
  - 3. The heap, from which memory is allocated
  - 4. The stacks used by each thread
  - 5. The standard C library

?fragmentation



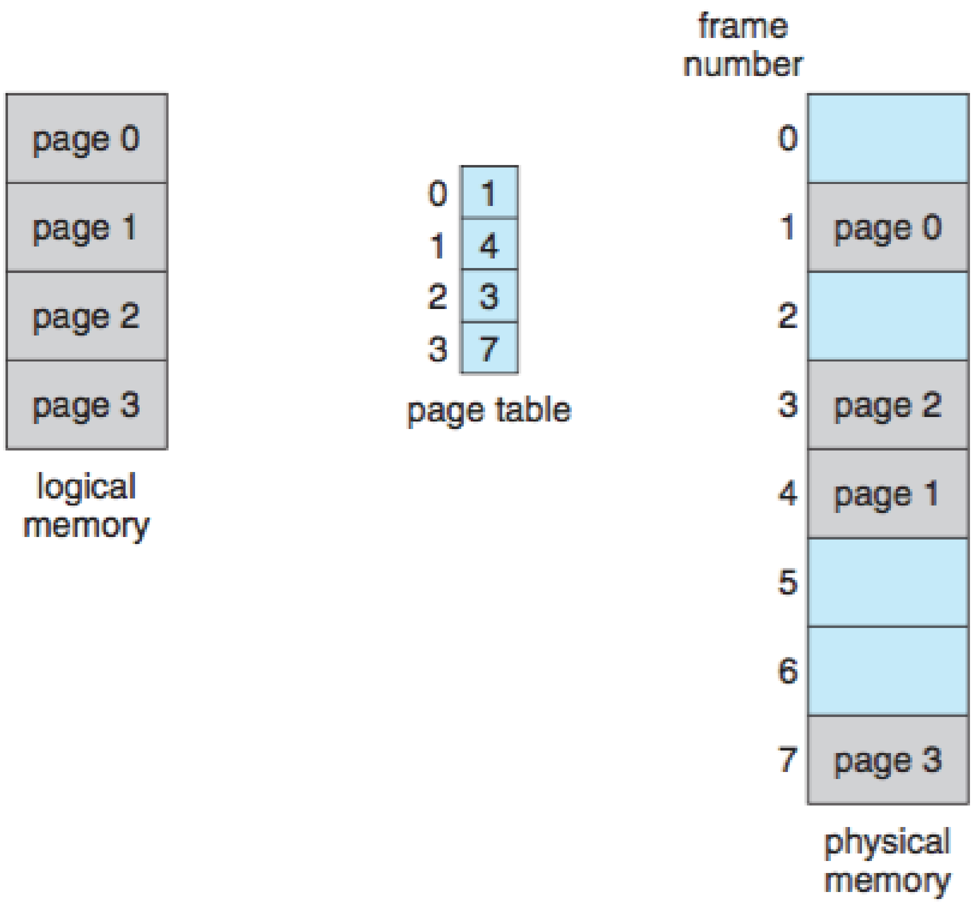
Programmer's view of a program

Segmentation



Segmentation
hardware

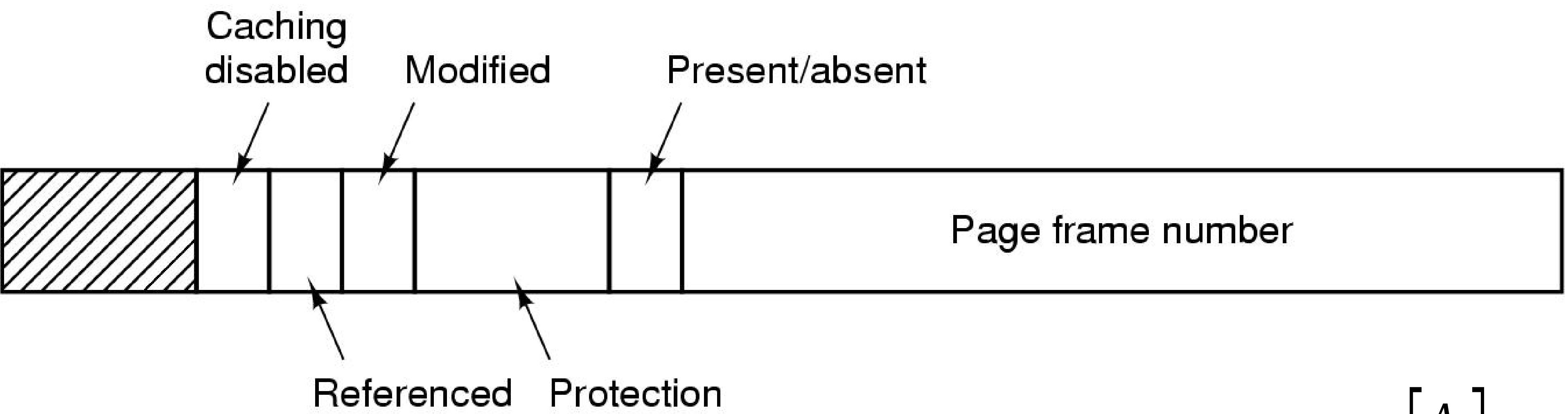
Paging 分页



- Page
- Frame
- Page Table

Paging model of logical and physical memory

Paging: page table



$$PageNum = INT\left[\frac{A}{L}\right]$$

$$offset = [A]\ MOD\ L$$

A: Logical Address

L: Page Size

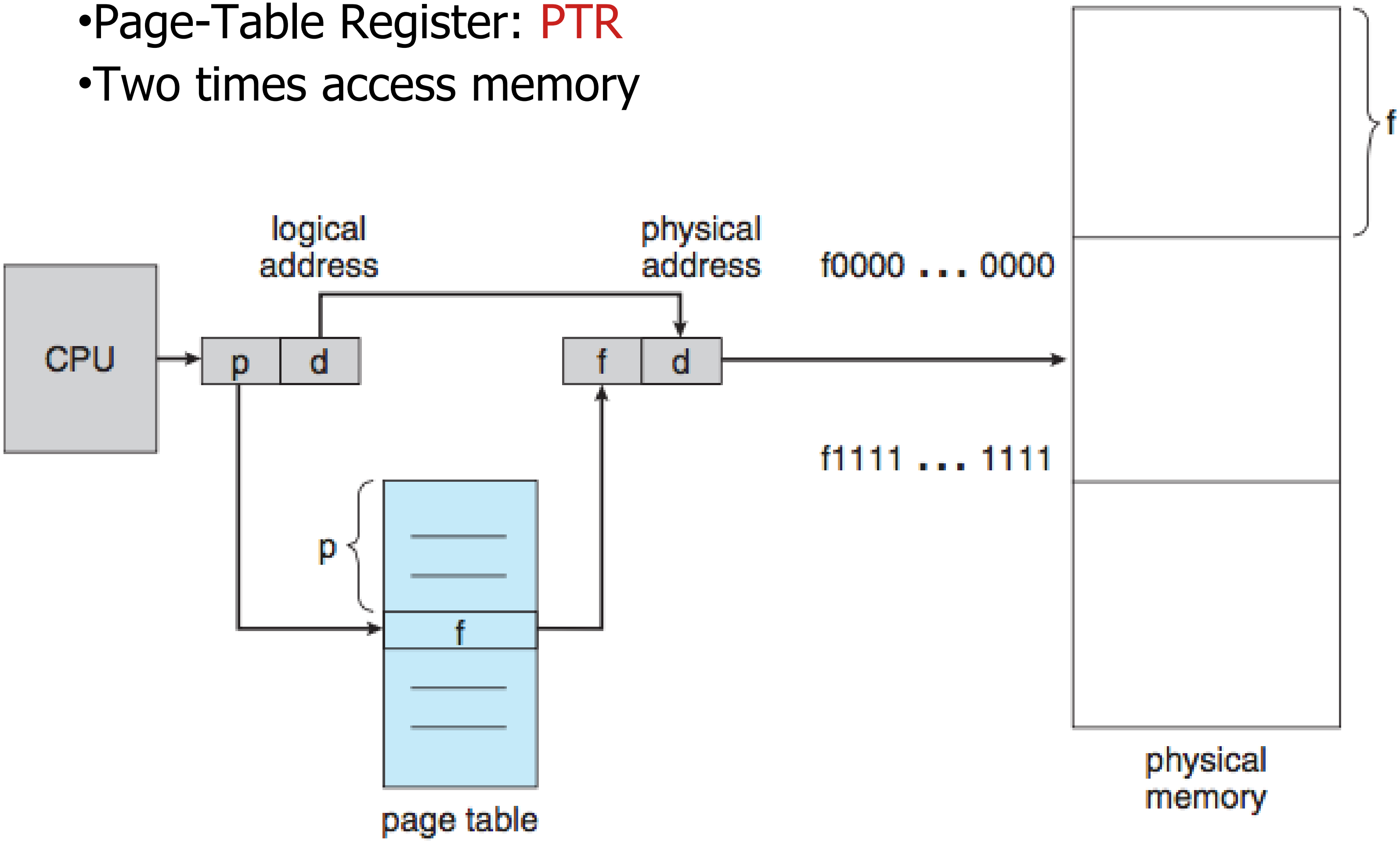
Address= (Page number, offset)

One Tuple

?fragmentation

Paging: hardware

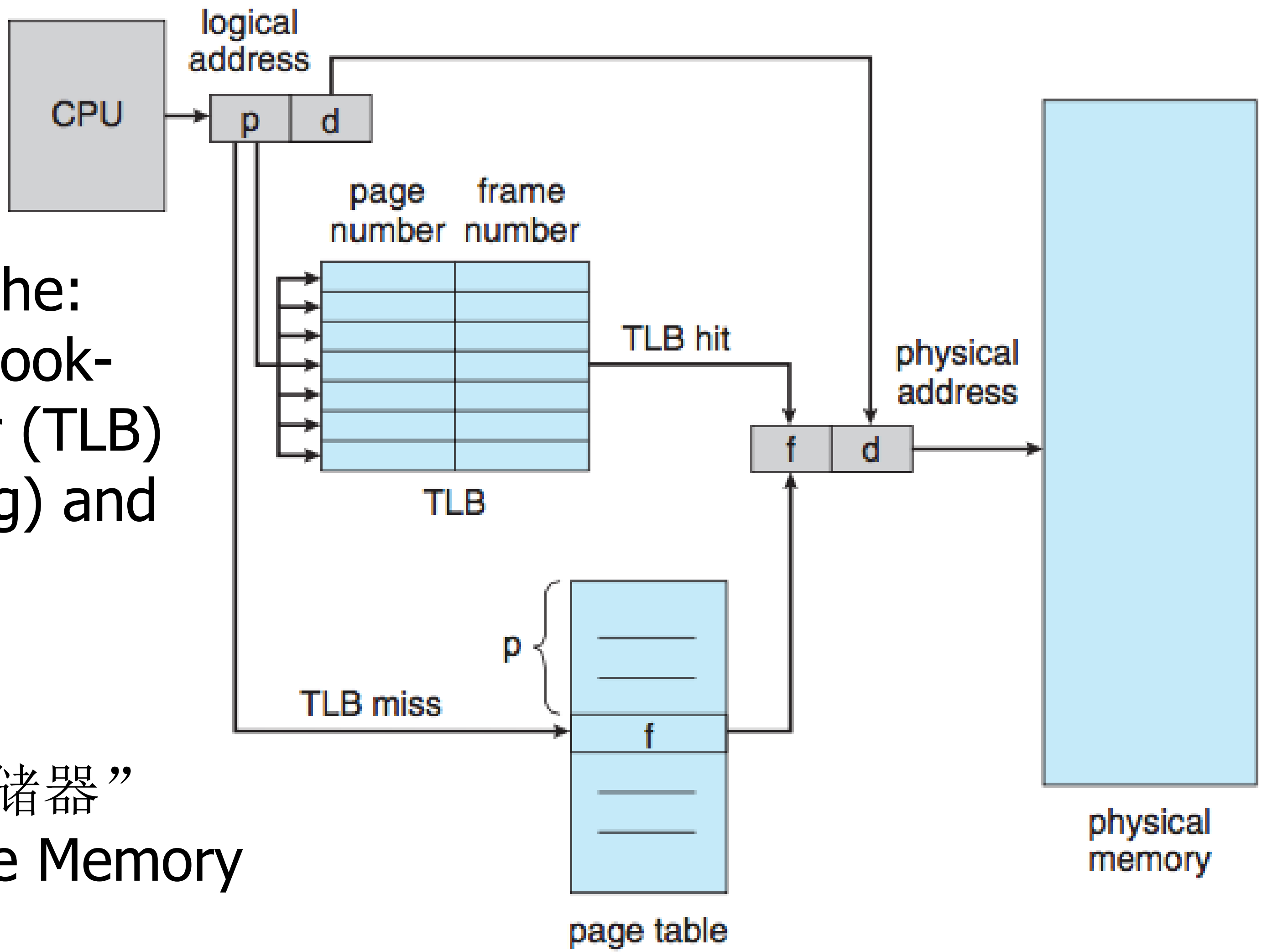
- Page-Table Register: PTR
- Two times access memory



Paging: hardware with TLB

hardware cache:
- translation look-aside buffer (TLB)
  - key (or tag) and value

- 快表 *,
- “ 联想存储器”
  Associative Memory



快表

- 在操作系统中，为了提高系统的存取速度，在地址映射机制中增加一个小容量的联想寄存器，即快表，用来存放当前访问最频繁的少数活动页面的页号。
  - 当某用户需要存取数据时，根据数据所在的逻辑页号在快表中找到其对应的内存块号，再联系页内地址，形成物理地址。
  - 如果在快表中没有相应的逻辑页号，则地址映射仍可以通过内存中的页表进行，得到空闲块号后须将该块号填入快表的空闲区中。
  - 如果快表中没有空闲块，则根据淘汰算法淘汰某一行，再填入新的页号和块号。
  - 快表查找内存块的物理地址消耗的时间大大降低了，使得系统效率得到了极大的提高。

CPU 中的缓存

- 典型案例是高速缓冲存储器 Cache
- 随着计算机硬件的发展， CPU 的执行速度越来越快，系统架构越来越先进，而主存的结构和存取速度改进则较慢，因此高速缓存技术将越来越重要。
- 高速缓冲存储器 Cache 是位于 CPU 与内存之间的临时存储器，它的容量比内存小但交换速度快。在 Cache 中的数据是内存中的一小部分，但这一小部分是短时间内 CPU 即将访问的。当 CPU 调用大量数据时，就可避开内存直接从 Cache 中调用，从而加快读取速度。

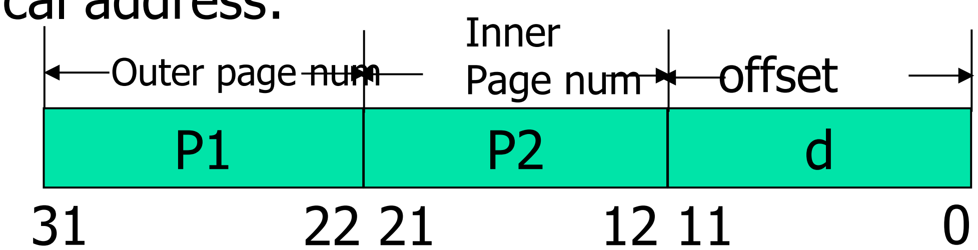
Paging: hardware with TLB

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

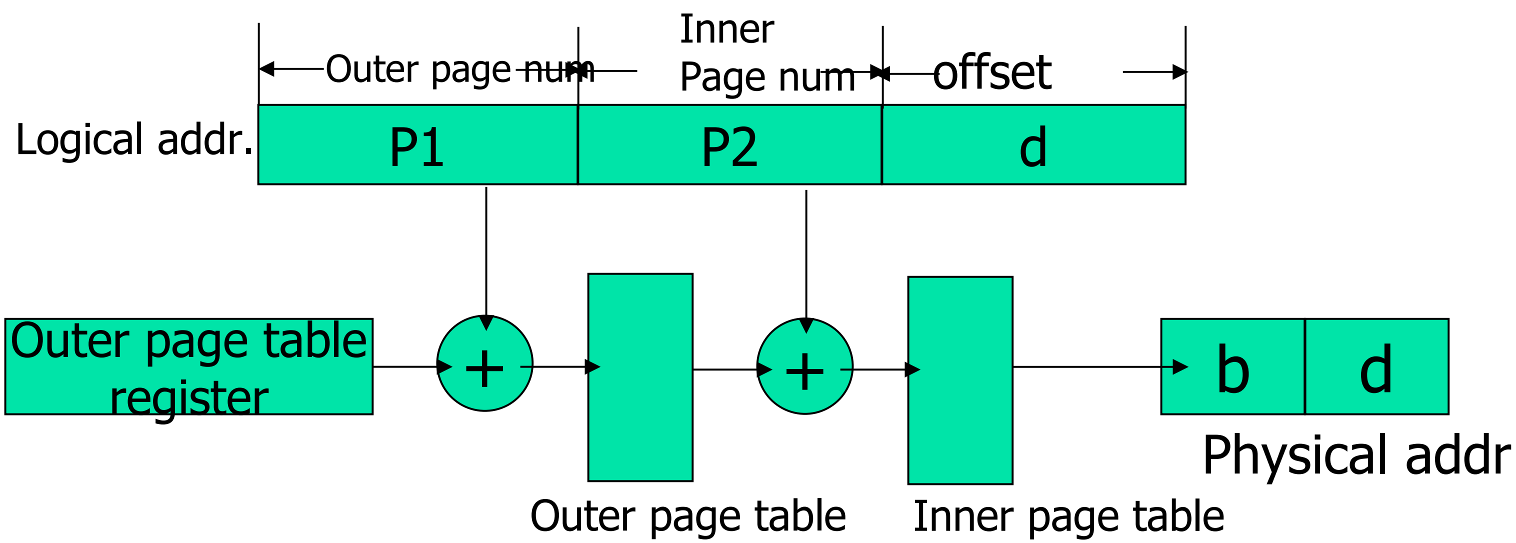
A TLB to speed up paging

Paging: Muti-level Page Table

Logical address:

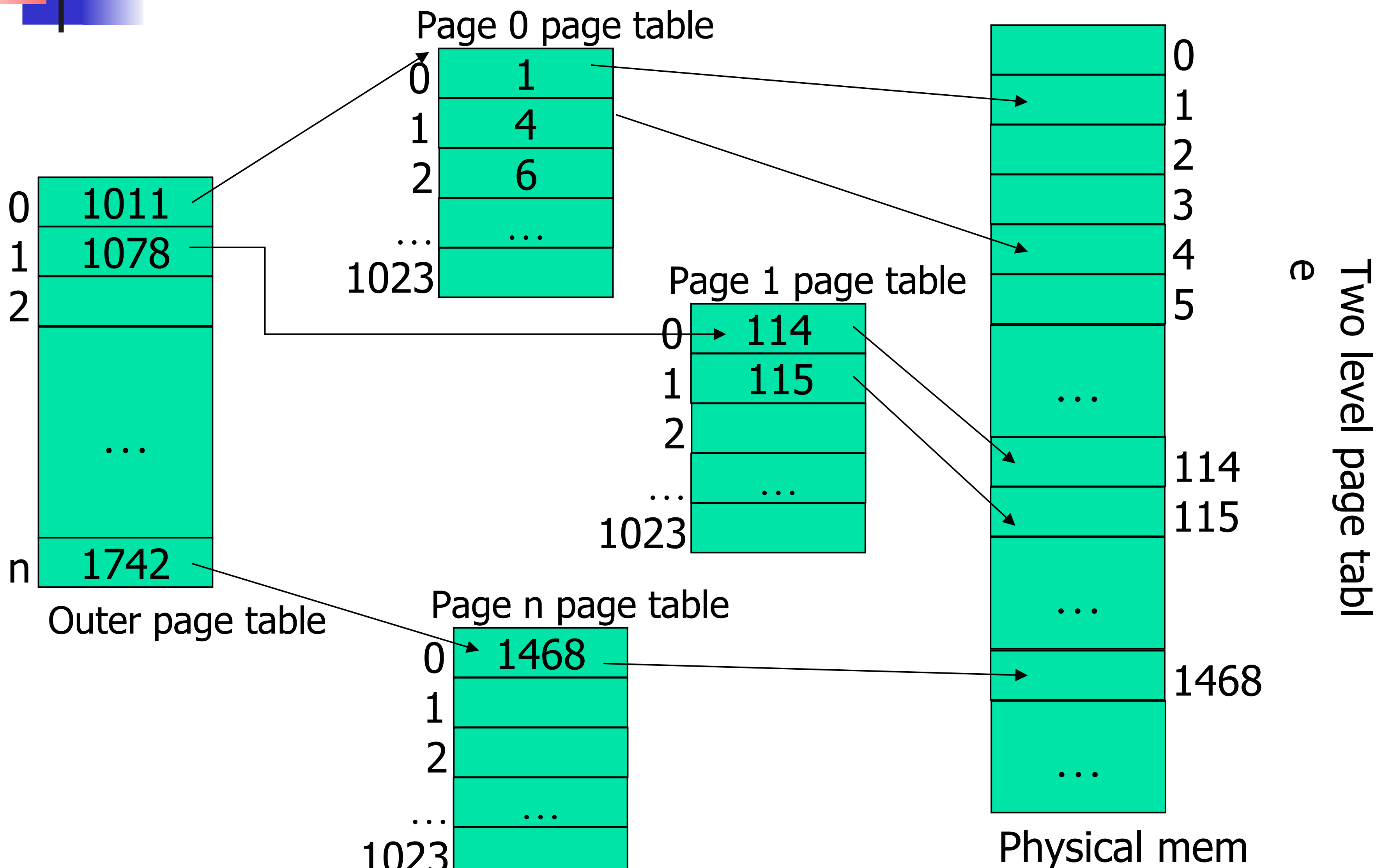


外层页表
Outer Page Table
Page Directory



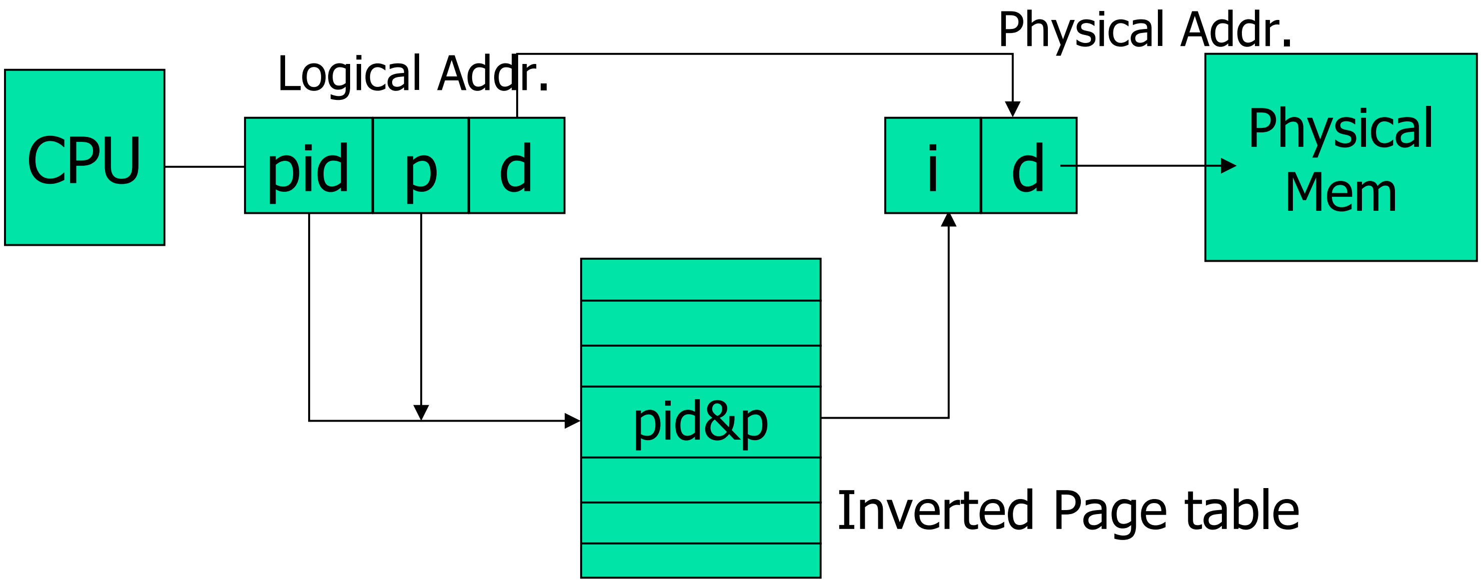
Two level page table

Paging: Muti-level Page Table

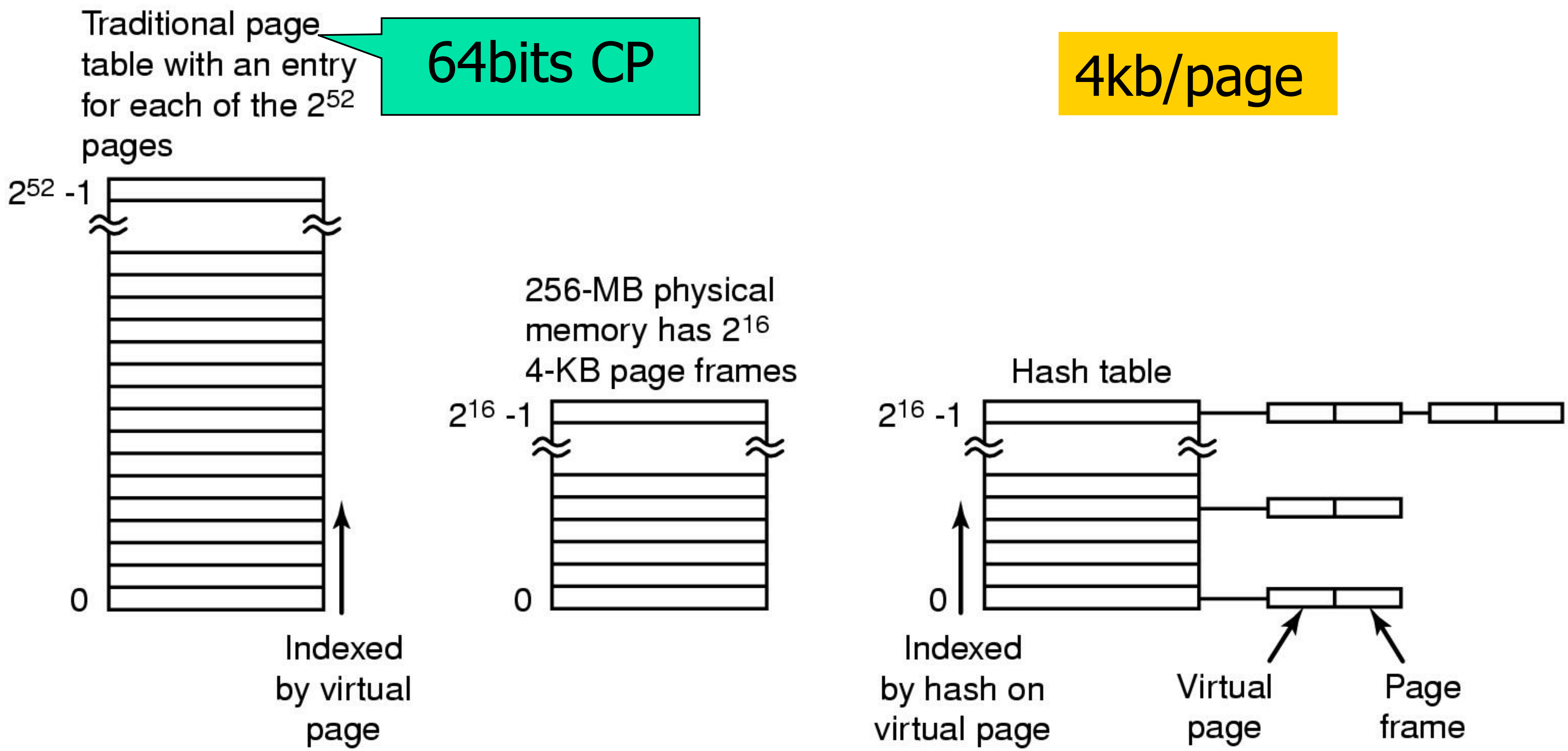


Inverted Page Tables 倒排页表

<process-id, page-number, offset>

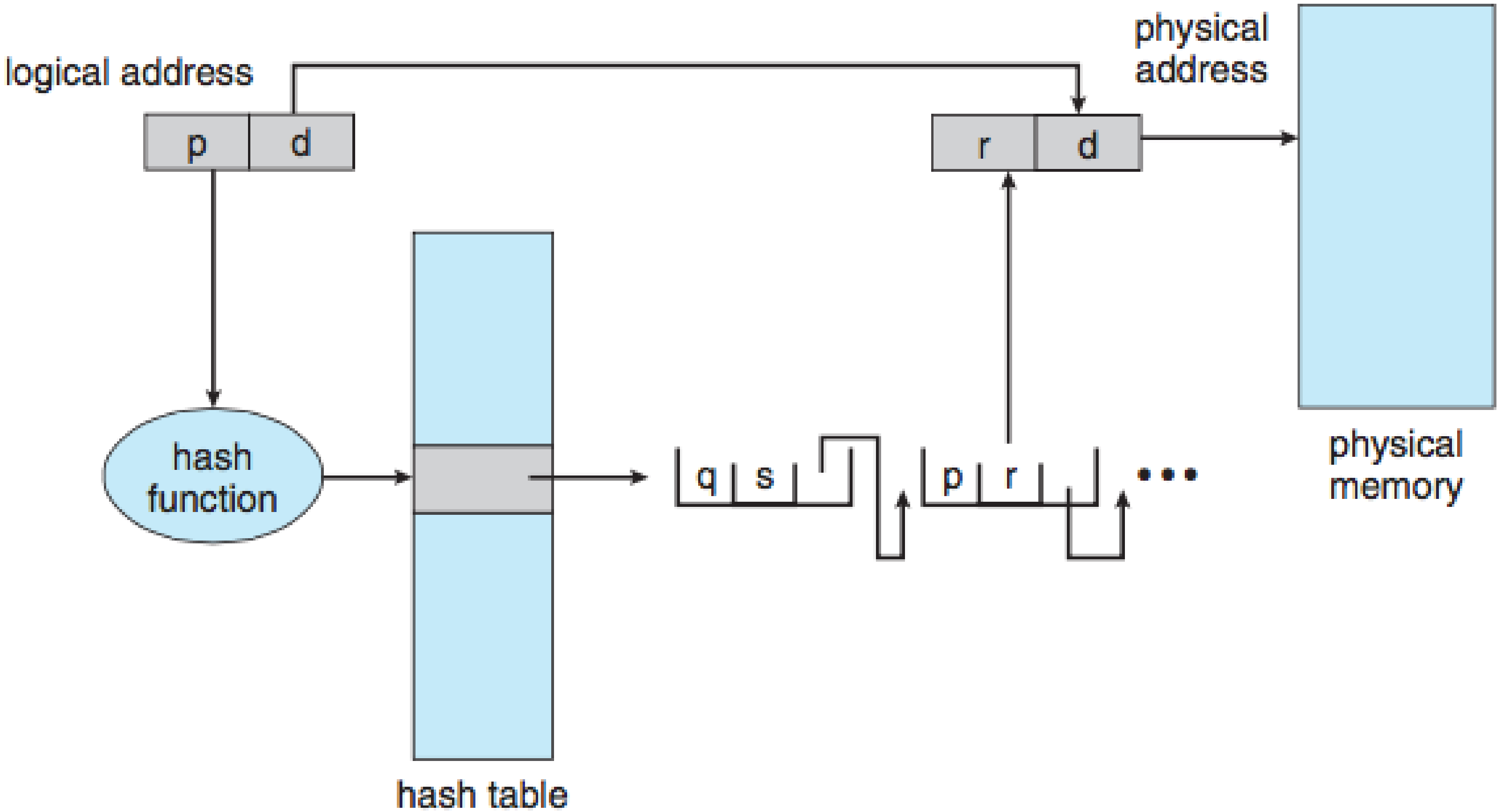


Inverted Page Tables



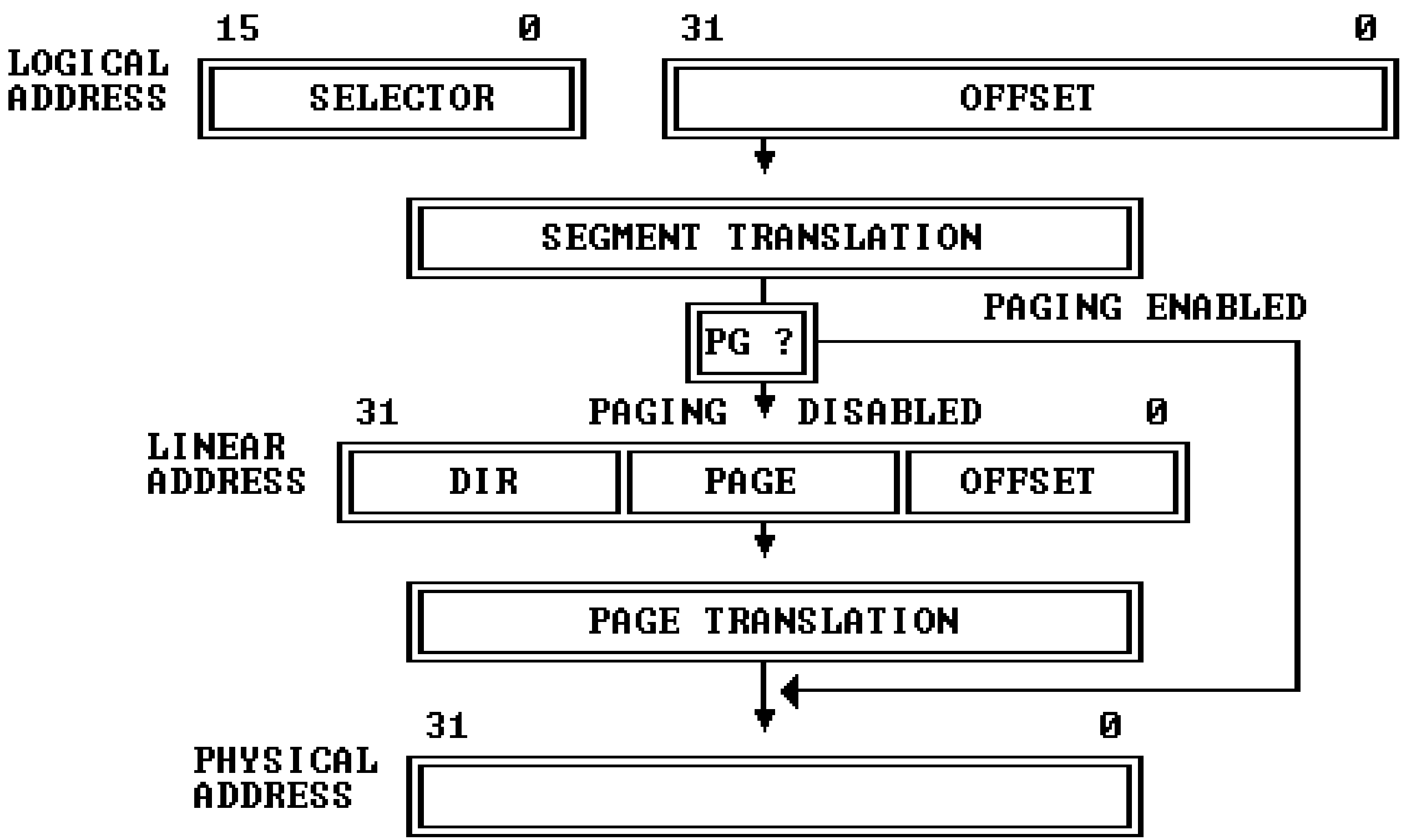
Comparison of a traditional page table with an inverted page table

Hashed Page Tables

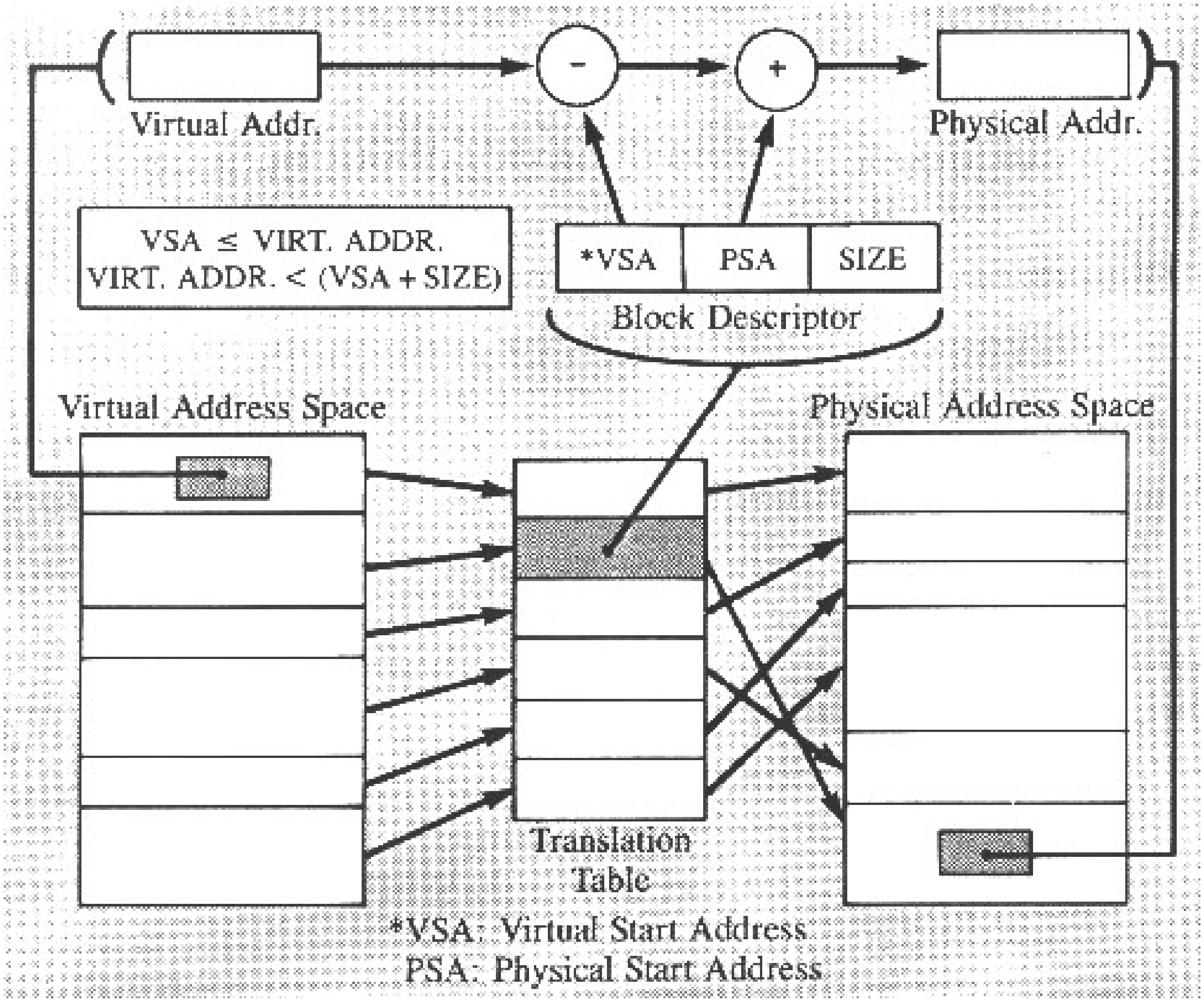


CURR MEMORY TRANSLATION

Logical Addr. -> (Virutal/)Linear Addr. -> Physical Addr.



CURR MEMORY TRANSLATION



Summary

- Memory Partitioning
- Swapping
- Segmentation
- Paging

Q&A?


рахмат
danke
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
mersi
kia ora
barka
welalin
spas
tack
спасибо
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
kiitos
dankie
nanni
nandri
dhanyavad
thank you
gracias
хвала
asante
manana
obrigada
tenki
bedankt
bayarlalaa
gràcie
hvala
mauruuru
köszönöm
akun dankon aciū
djiere dieuf
tau
mochchakkeram
murakoze
enkosi
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
дякую
mamnun
go raibh maith agat
chokrane
sobodi
dekuji
obrigado
sagolun
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
rahmet
terima kasih
kam sah hamnida
rahmat
diolch
dhanyavadagalu
shukriya
merce
мерси
감사합니다
xièxie
ευχαριστώ
merci