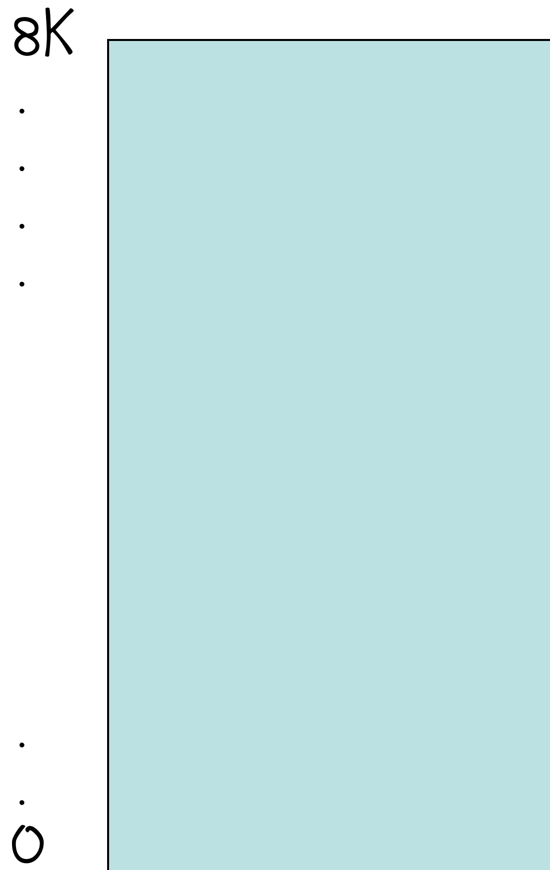
Memory Management

CS 447

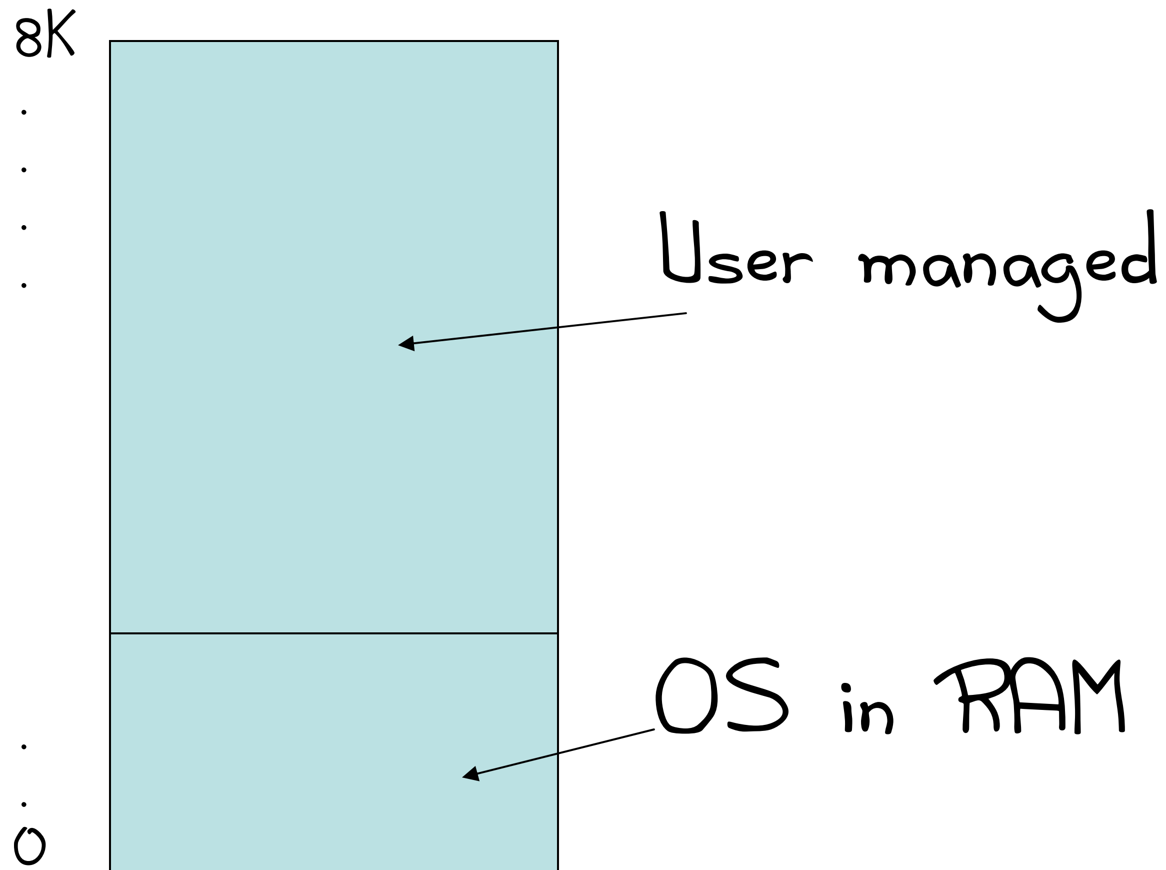
Prof. R.K. Joshi
Dept of CSE
IIT Bombay

Some Simple Memory schemes

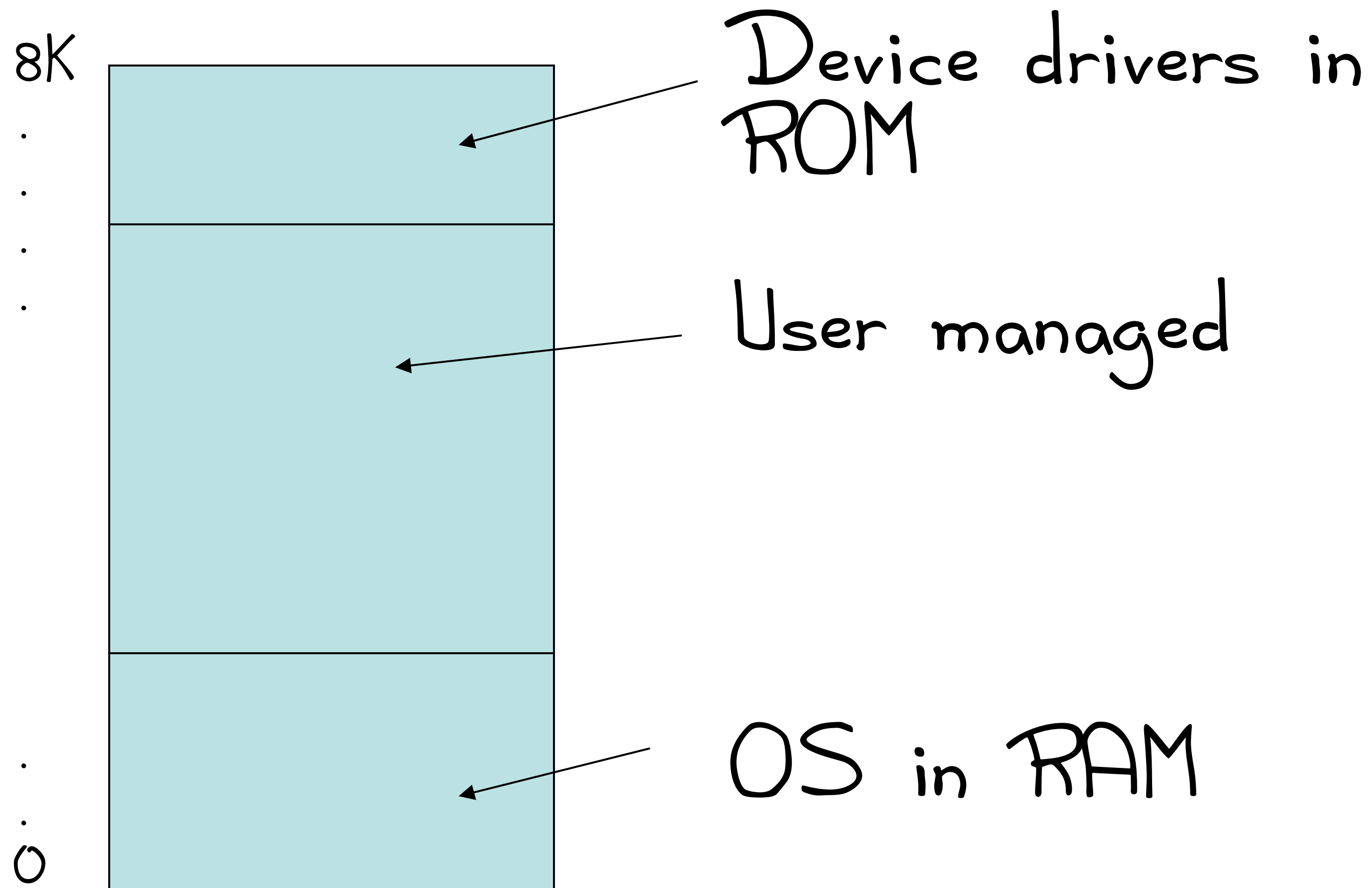


User managed

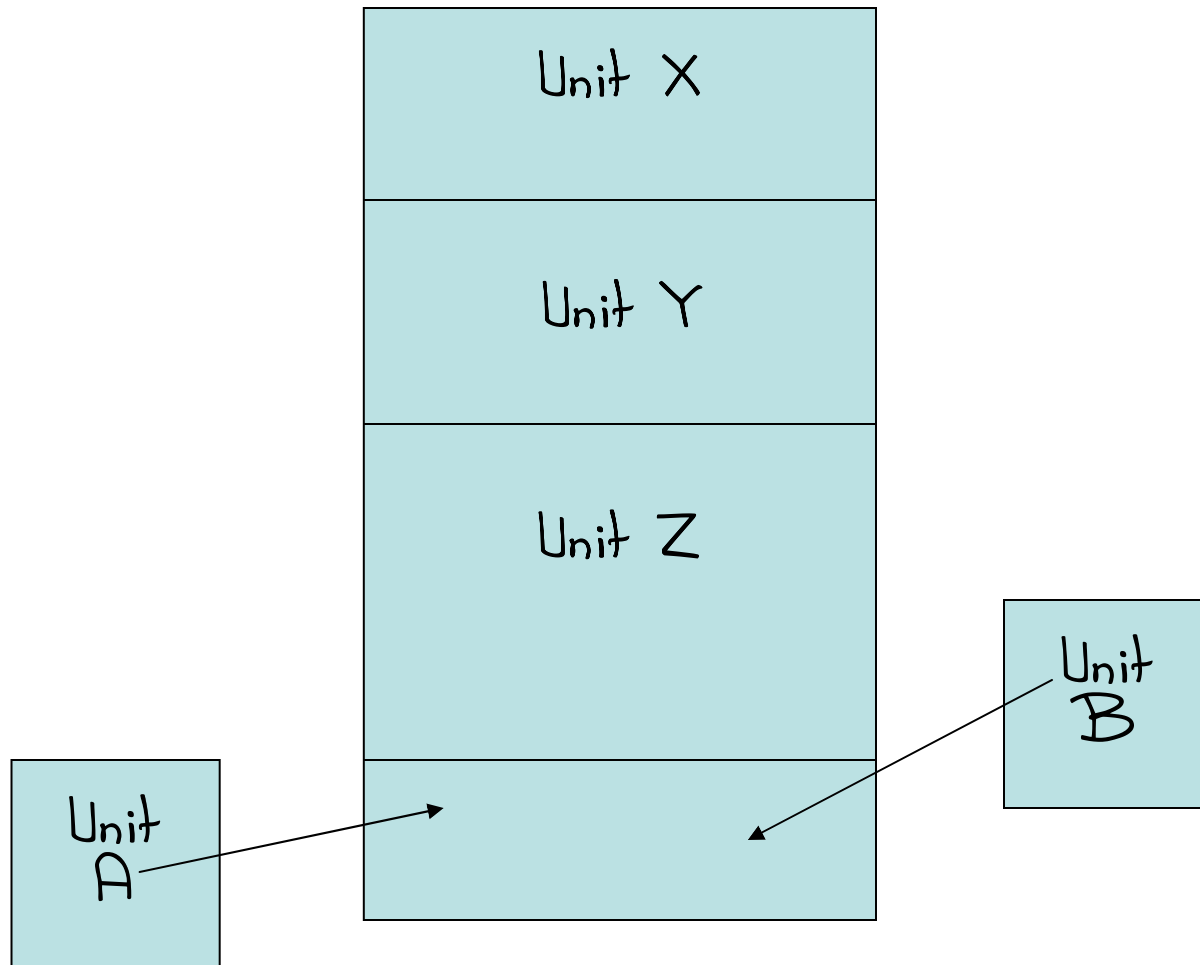
Some Simple Memory schemes



Some Simple Memory schemes



Overlays: User level memory management (e.g. TurboPascal)



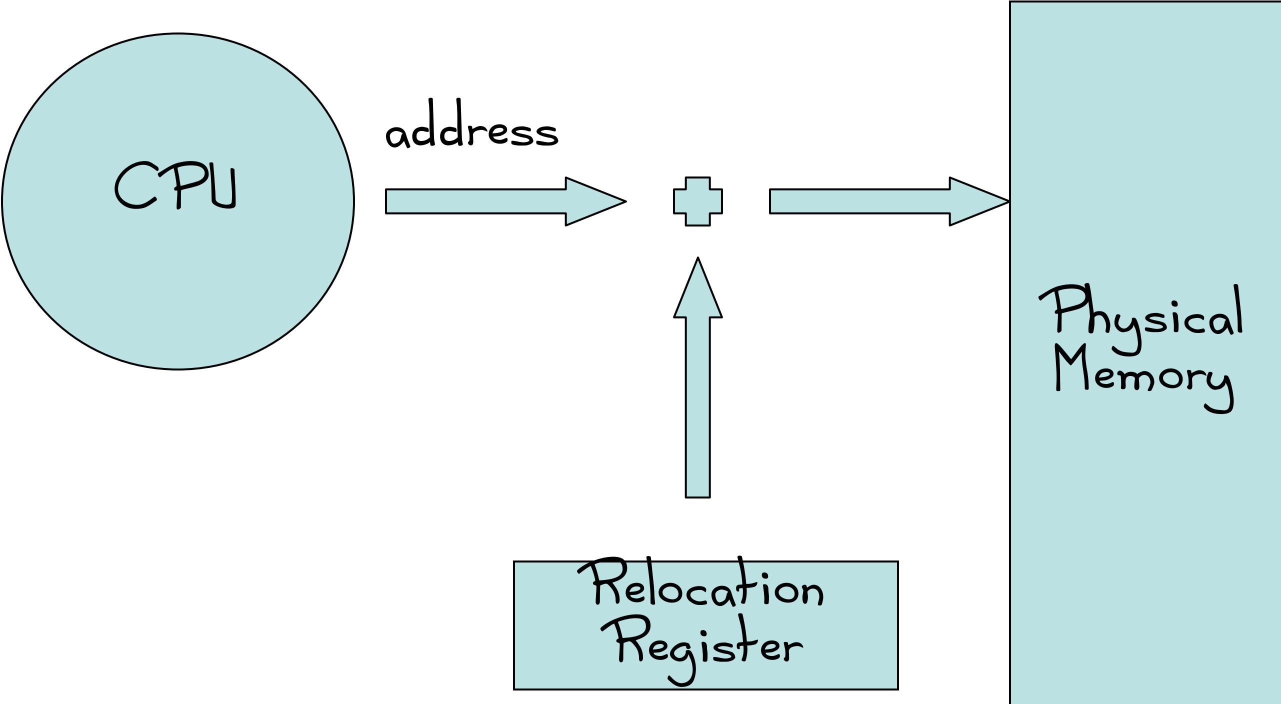
When is Address Binding performed?

- Compile Time (Absolute Addressing)
- Loading Time (Relocatable code located at load time)
- Execution Time (Code relocatable throughout execution)

Linking

- Static linking
- Dynamic Linking
  - Implications to memory?

Logical vs physical address space

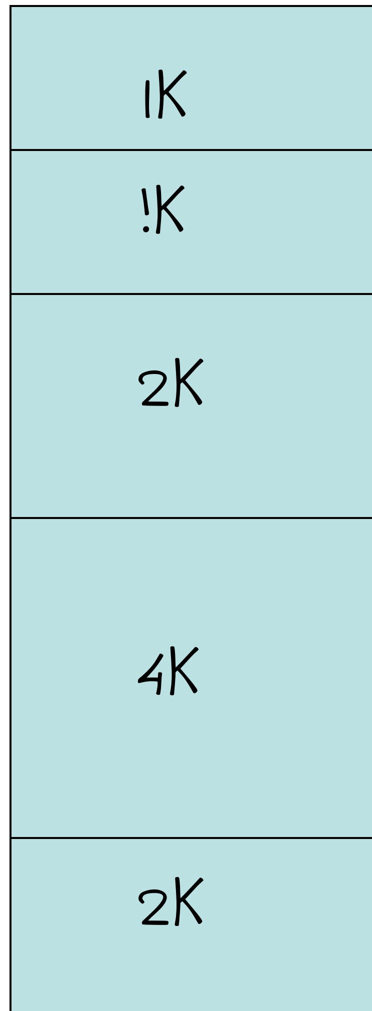


Memory Allocation: Continuity and chunk size

- Contiguous
- Non-contiguous

- Fixed Partition Size
- Variable Partition Size

Contiguous allocation, fixed partitions

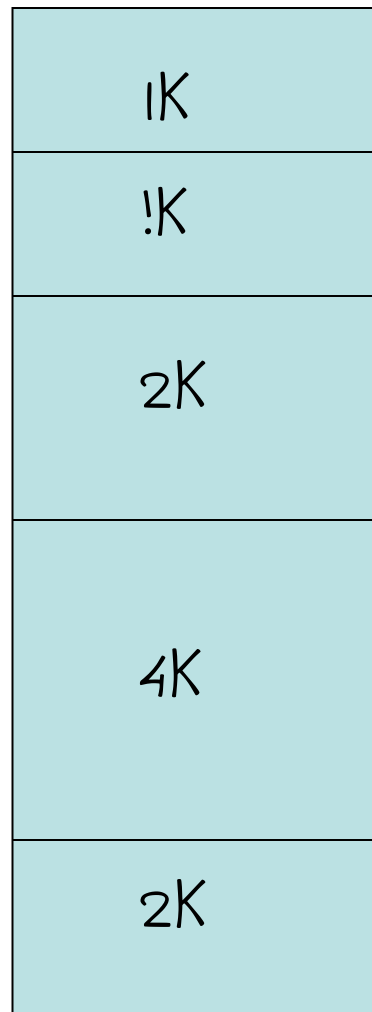


Job arrivals:

2K,2K,1K,1K.......

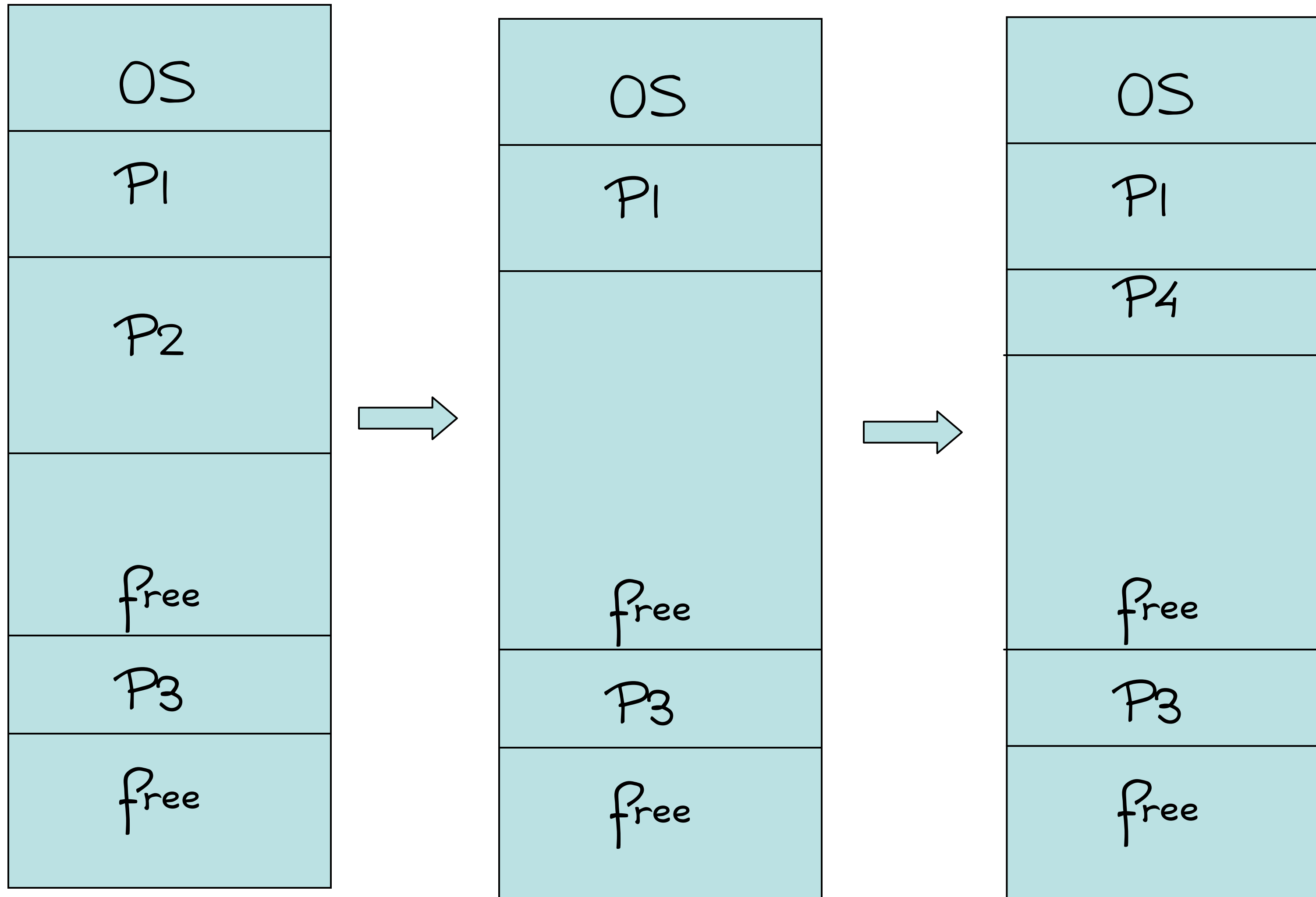
Multiprogramming with variable partitions

Allocation policies

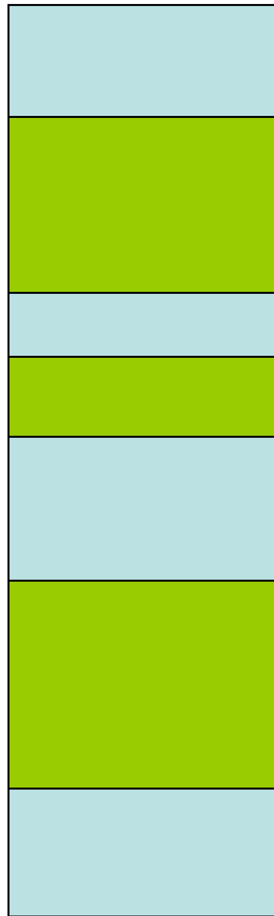


Typically Separate queues for partitions

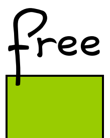
Contiguous Allocation, Variable Partition



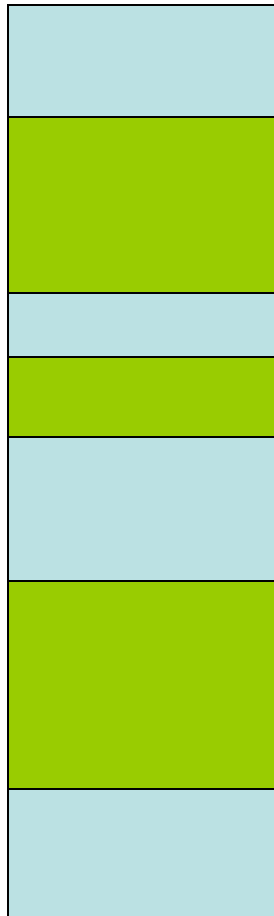
Allocation Policies



Which partition should be allocated for a given request of size Z?



Allocation Policies

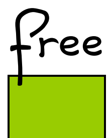


First Fit

Best Fit

Worst Fit

What data structures are used to maintain Free space lists?h



Fragmentation

- External

  - Occurs in variable partitions

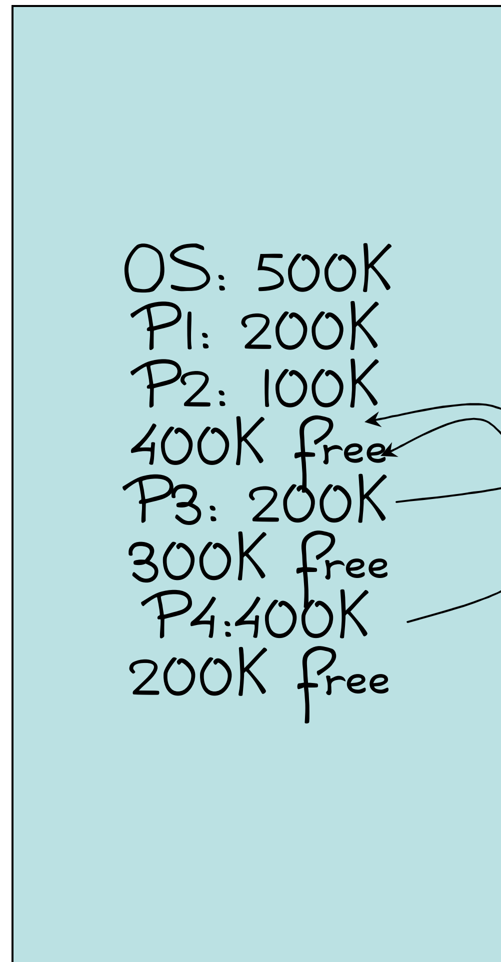
- Internal

  - Occurs in Fixed partitions

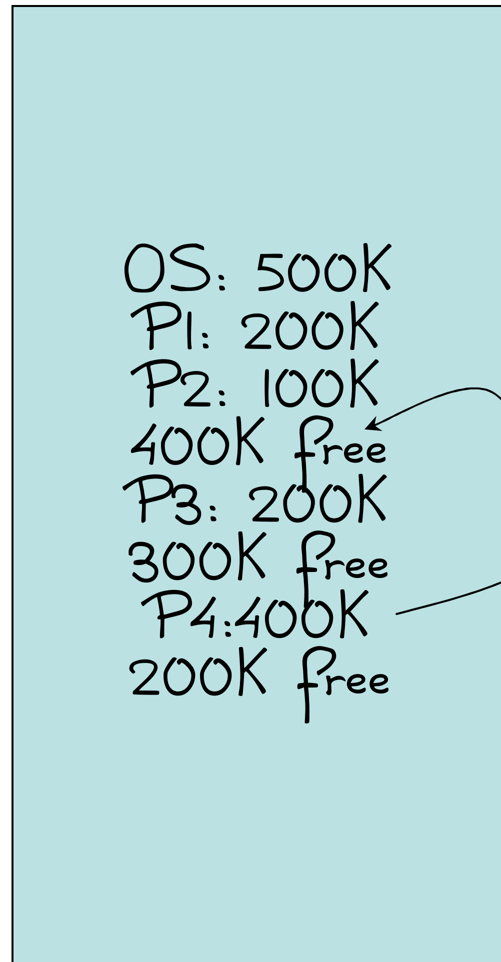
Compaction

OS: 500K
P1: 200K
P2: 100K
400K free
P3: 200K
300K free
P4:400K
200K free

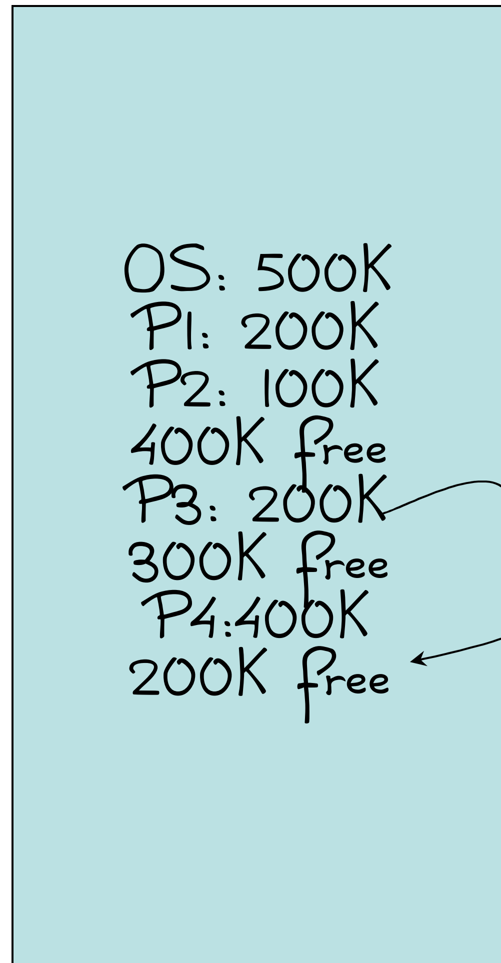
Compaction



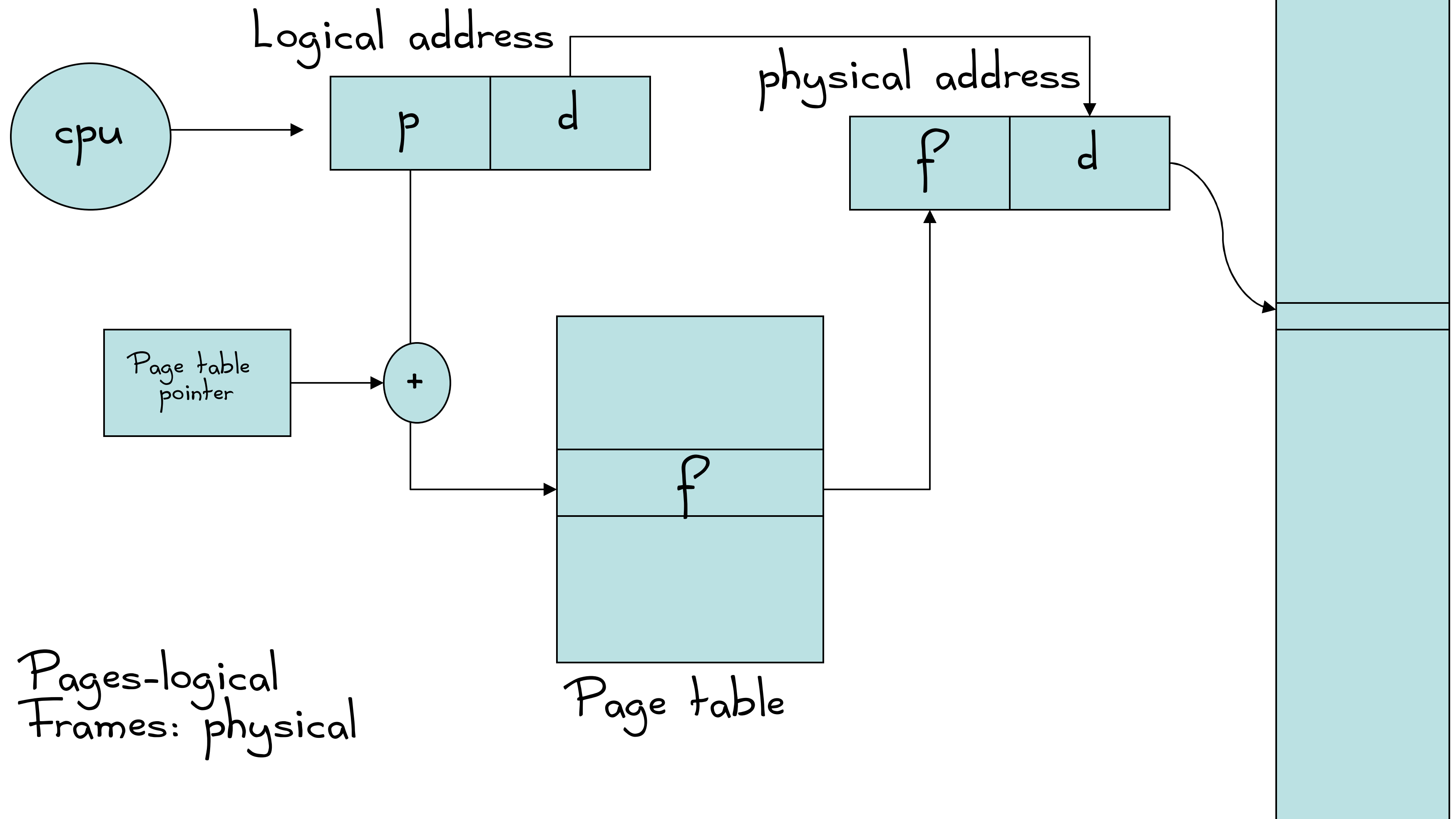
Compaction



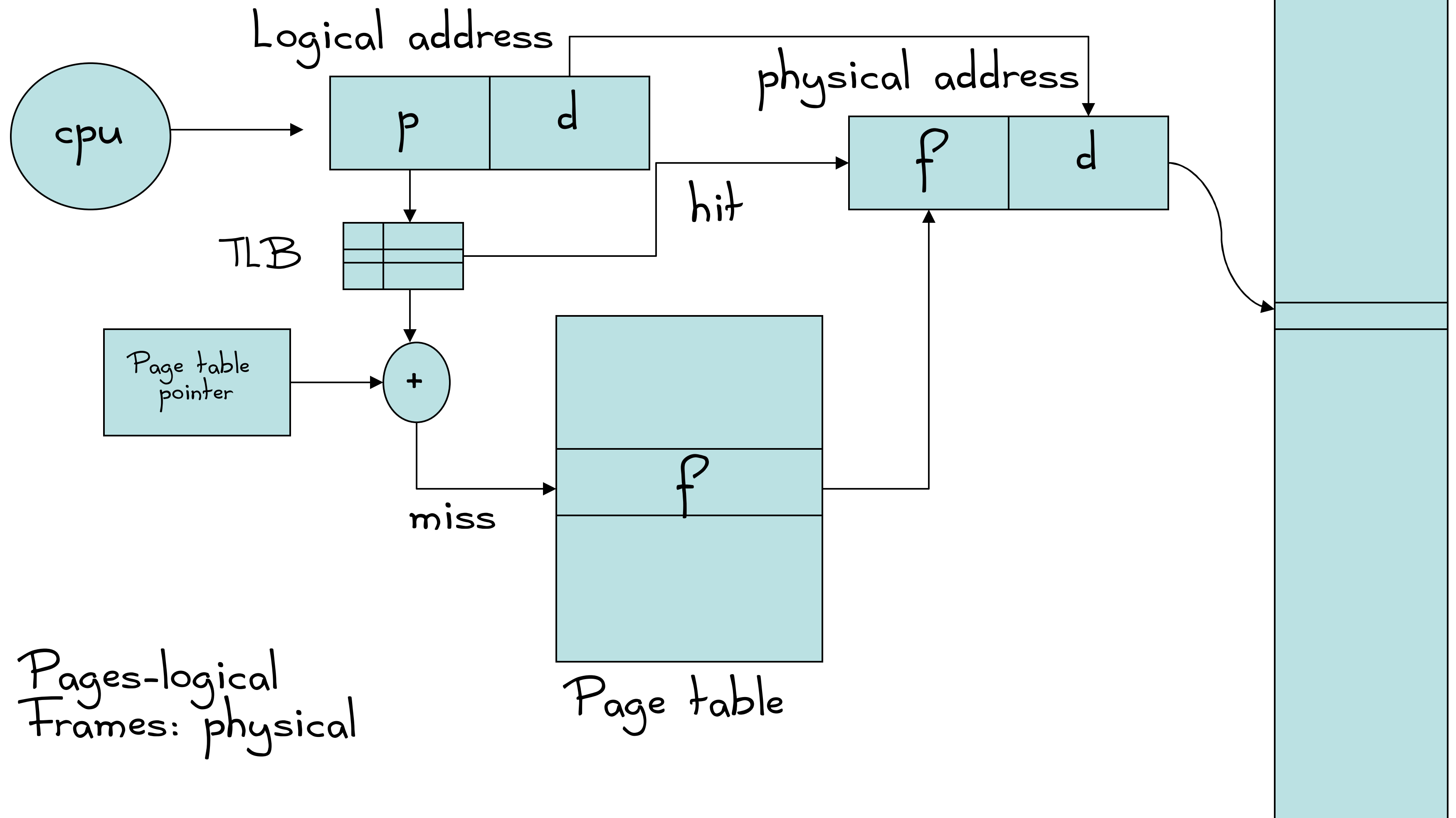
Compaction



Non-contiguous allocation fixed partition: Paging



Translation look ahead cache (TLB)



TLB performance

- Hit ration: 80%
- Prob. Of TLB hit: .8
- tLB access time: 20 ns
- Mem access time: 100ns

Calculate effective access time?

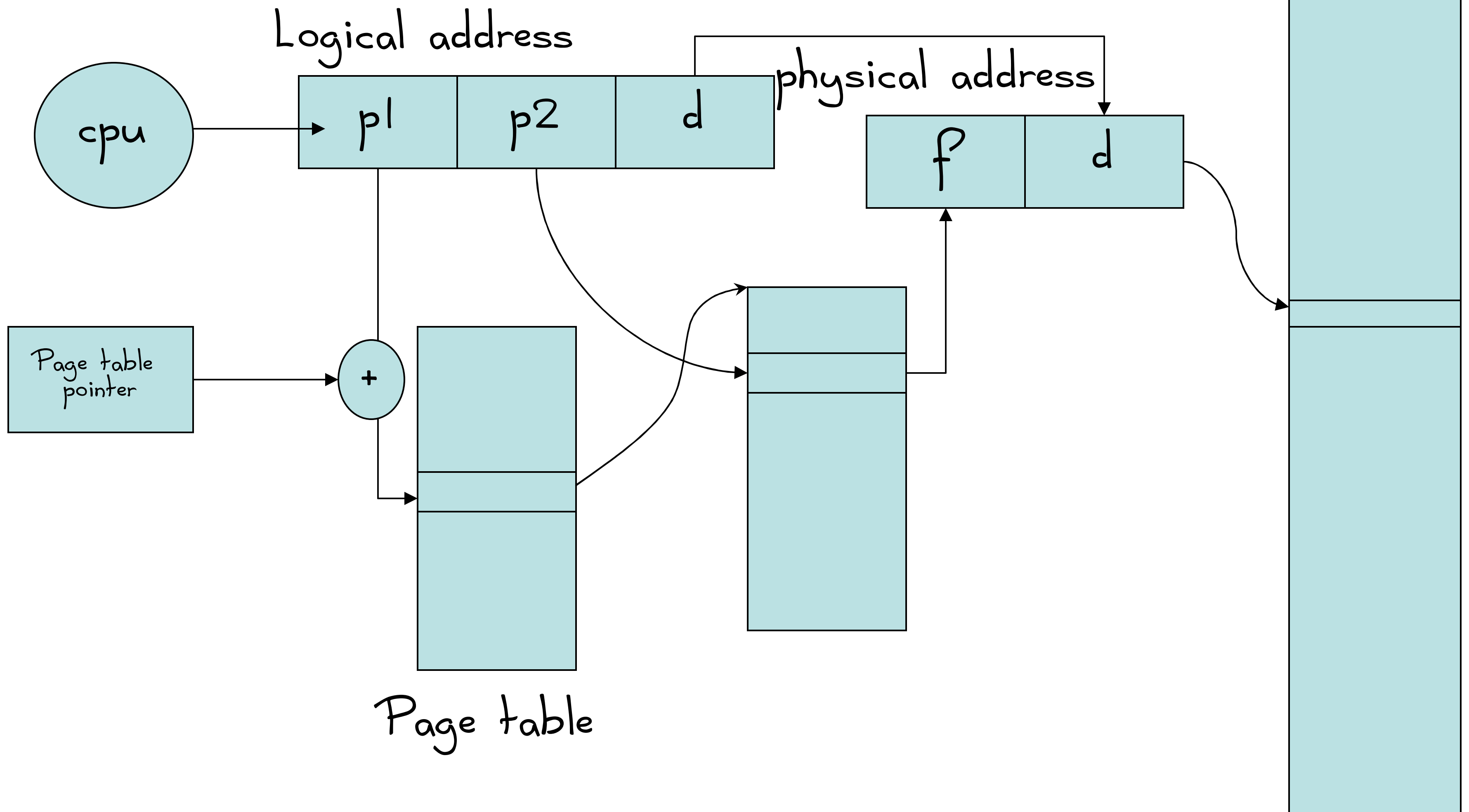
Large address spaces

- What about large address spaces?

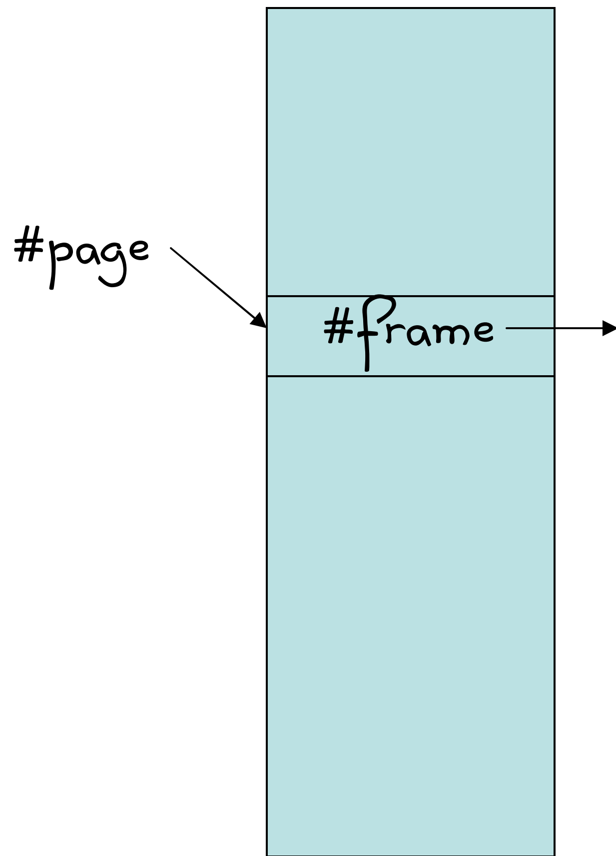
- 32 bit space: page size 4K,
- How many entries in PT?

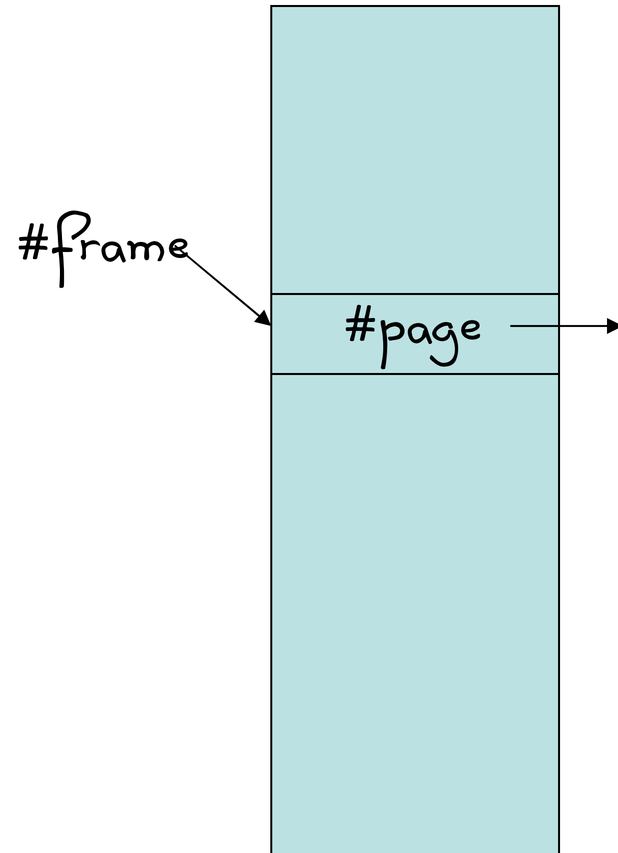
- Size of PT?

Multilevel Paging



Inverted Page Table



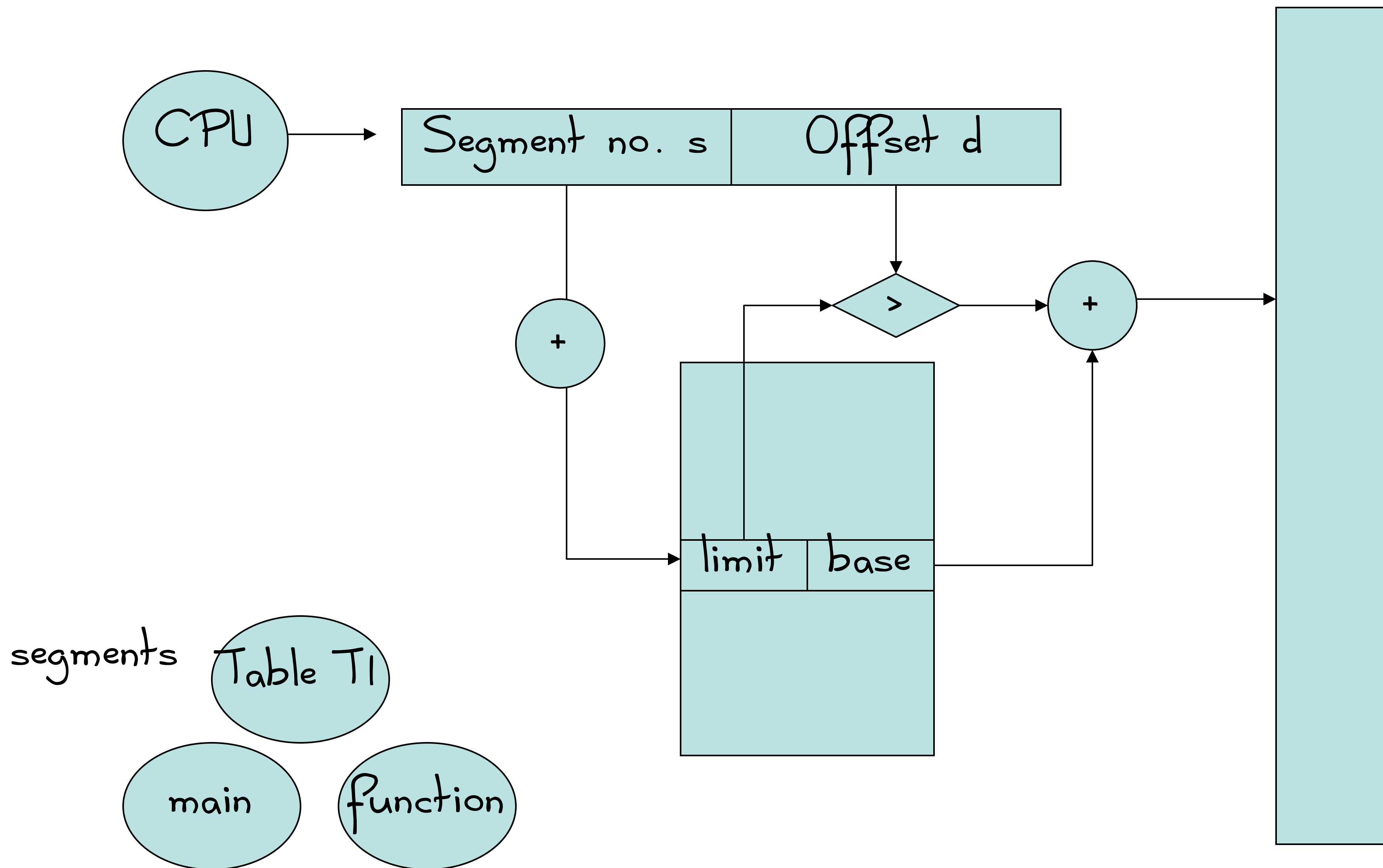


Inverted Page Table

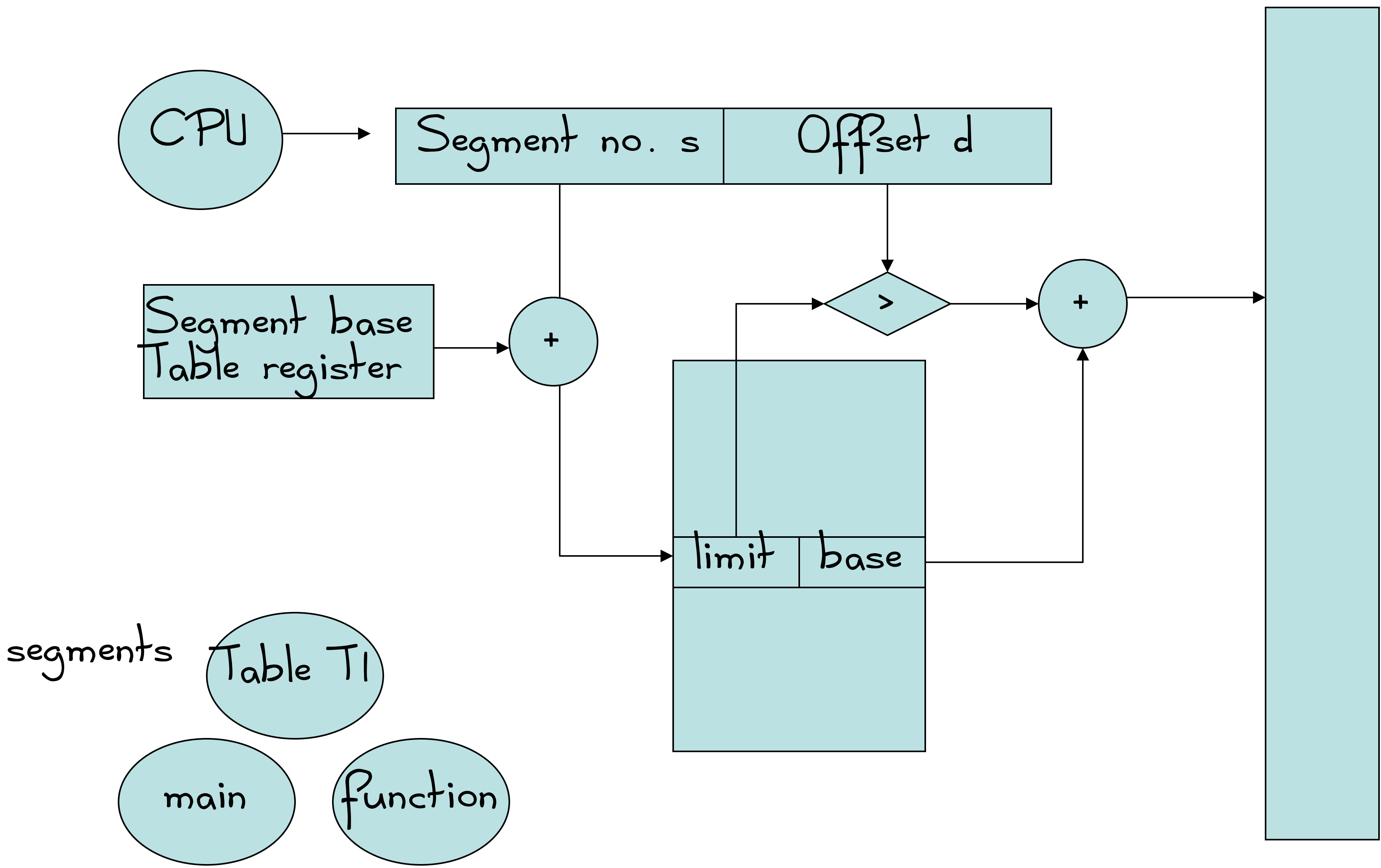
- Does it need more information?
  - Tuple stored: pid, page no, frame no
  - Hash on: pid, page no.

- 64 bit address space, 4KB page size, PT size= ??
- Inverted PT size= for 1GB RAM, 4KB page/frame size:

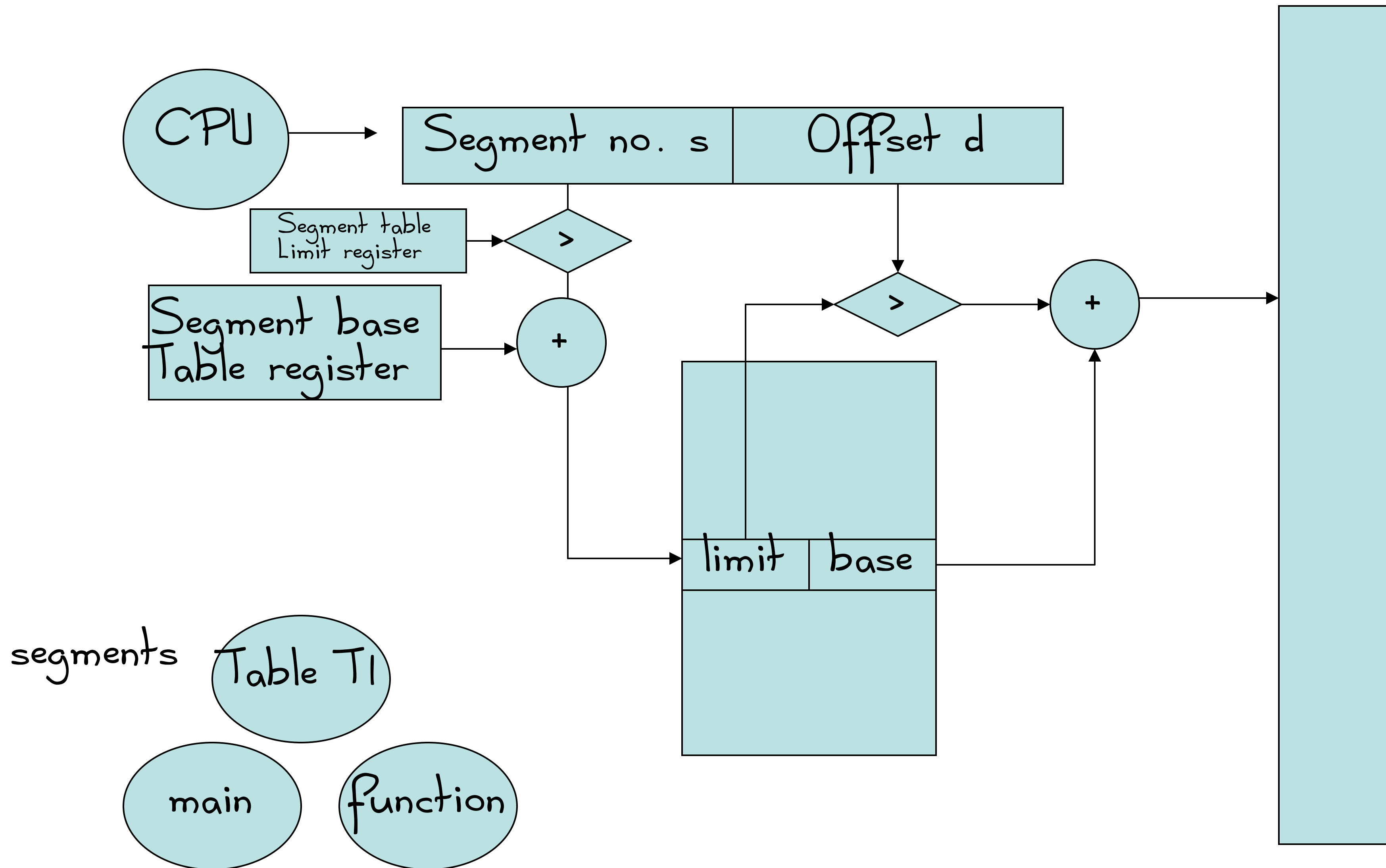
Non contiguous allocation and variable partition sizes: Segments



STBR



STLR



Paged segmentation vs segmented paging

- Page the segment table
- Page every segment

Page the segments

segment	offset	
	page	offset

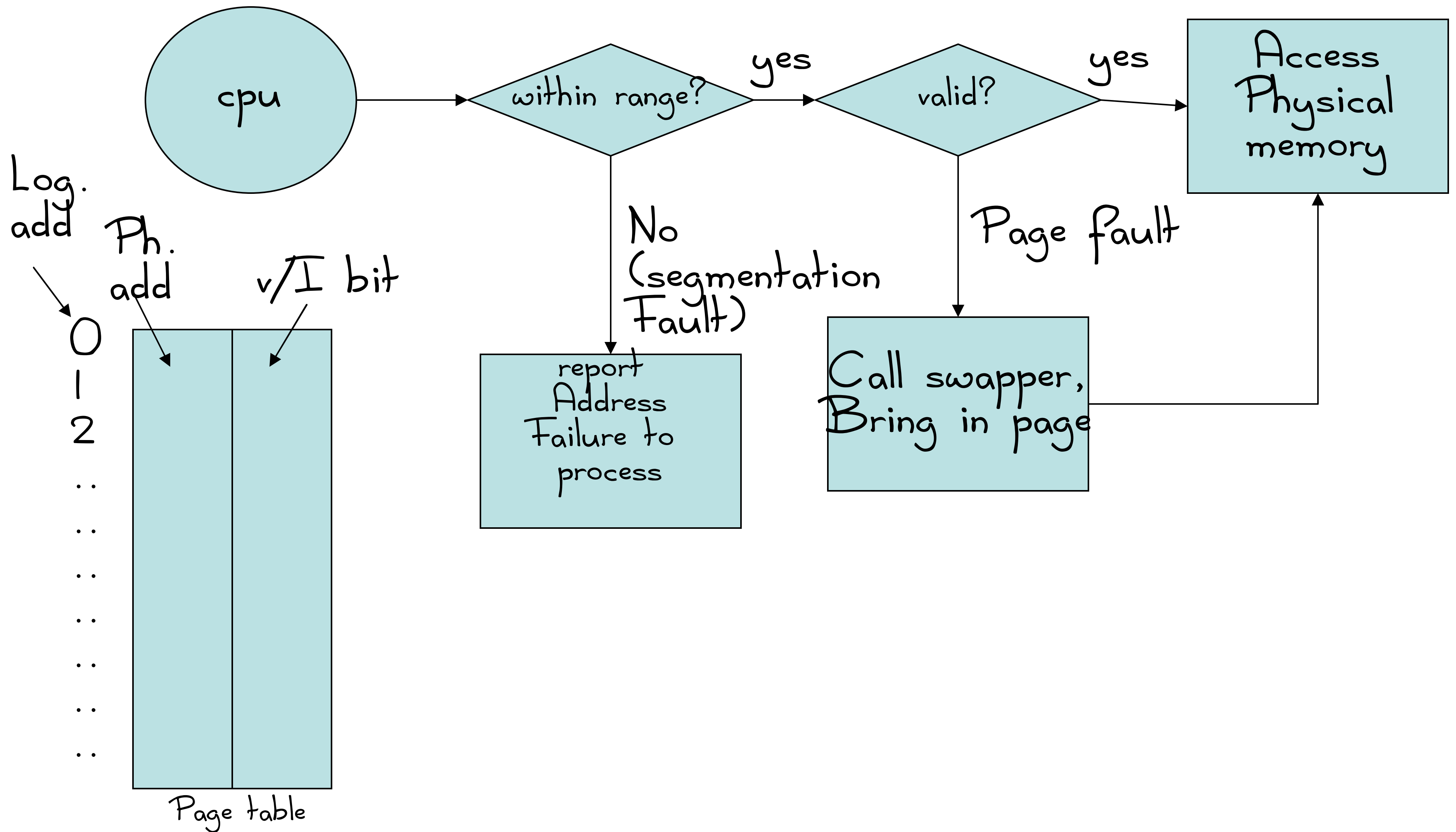
Page the segment Table

<table>
<tr><td colspan="2">segment</td><td>offset</td></tr>
<tr><td>page</td><td>offset</td><td></td></tr>
</table>

Protection against invalid addresses

- PTLR
- STLR/segment limt
- Valid/invalid bit → for swapped pages

Swapping



Demand paging

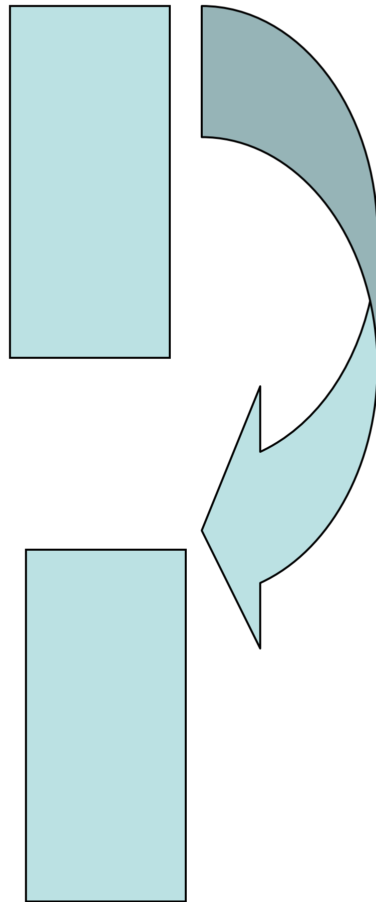
- Bring in a page only when it is needed
- Pure demand paging: initially nothing is loaded

- What’s the minimum no. of pages that needs to be loaded to complete an instruction?

Example instructions

- Add A, B, C *(e.g. C=A+B)*
  - Fetch opcode ADD and decode it
  - Fetch A instruction
  - Fetch B
  - Fetch C
  - Fetch *A
  - Fetch *B
  - Add
  - Store sum at C
- Page fault at any step, entire instruction is restarted

An MVC instruction



Move blocks of memory

Will a simple restart of instruction on page fault work?

2 solutions

- Save context and restore
- Check initially whether all needed addresses are in physical memory

Effective access time

- Ma= Memory access time:
- P = prob. Of page fault
- Effective access time:
  - p*(fault service time)+(1-p)ma
- Fault service time:
  - Trap to OS
  - Save registers
  - Determine that the intr is a page fault
  - Check that reference is legal and determine disk address
  - Issue disk read
  - Wait for device → another process may be scheduled
  - Begin transfer
  - disk i/o completion
  - Correct page table
  - Restore registers and
  - Start process again
  - Access physical memory

Page replacement Algorithms

To get lowest page fault rate

e.g. 100, 0432 , 0101, 0612, 0102, 0103, 0103, 0611, 0412

Reference string: 1,4,1,6,1,1,1,6,4

Page faults with 1 frame, 3 frames?

Example

- 7 0 1 2 0 3 0 4 2 3 0 3 1 2 2 0 1 7 0 1

- FIFO replacement

- Faults with 3 frames=??
- Faults with 4 frames=??

Another Example

- 1 2 3 4 1 2 5 1 2 3 4 5

- FIFO replacement

- #Faults with 3 frames=
- #Faults with 4 frames=

- Belady’s anomaly

Another Example

- 1 2 3 4 1 2 5 1 2 3 4 5

- FIFO replacement

- #Faults with 3 frames=
- #Faults with 4 frames=

- Belady’s anomaly

Cause of Belady’s anomaly

- Will Set of pages in memory for n frames be always subset of pages in memory with n+1 frames, then
  - LRU?
  - FIFO?

Global vs local page replacement

- Select from all processes
- Select form self/user

OPT

- Replace page that will not be used for longest period of time
- > requires knowledge of future
- Approximation:
  - Least recently used

Implementation of LRU

- Time stamp
- Approximation:
  - Reference register schemes
  - Stack (pull a page that is referred, and push on top) – bottom page is LRU page

Additional Reference bit algorithm

- 8 bit register
- 1 reference bit
  - Set when page is accessed

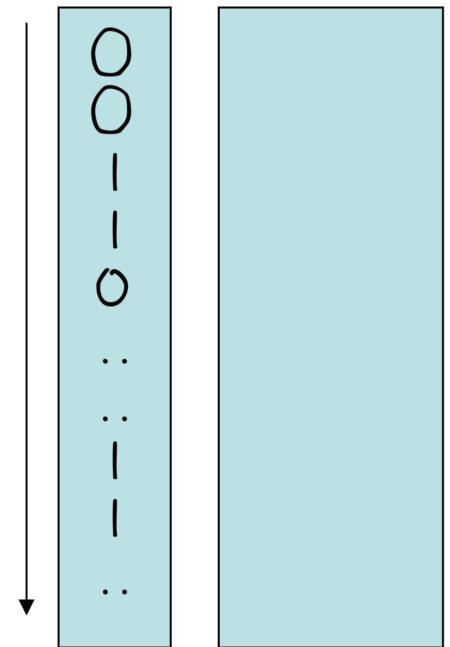
  - Move reference bit by right shift every 100 ms for all processes

  - Smallest number: LRU
  - Largest number: MRU

If you had only 1 bit?

Second chance algorithm

- When searching for a victim:
  - If ref bit=1, give it a chance, turn it to 0
  - If ref bit=0, replace it
  - Search in FIFO order



- If you had 2 bits?
  - A reference bit: set if page is referenced
  - A modify bit: set if page is modified

Enhanced Second chance algorithm

4 possibilities:

Ref bit	Modify bit
– 0	0
– 1	1
– 1	0
– 0	1

Ref bit modify bit

- 0 0
  - Neither recently used nor modified
  - Best to replace
- 0 1
  - Not recently used but is dirty
  - Not good to replace as page has to be written back
- 1 0
  - Recently used but is clean, may not be used again
- 1 1
  - Recently used and modified also

An Example

- Initially
  - Pg1: 0 0
  - Pg2: 0 0
  - Pg3: 0 0
  - Pg4: 0 0
- After read p1, write p2, write p3 →
  - 1 0
  - 1 1
  - 1 1
  - 0 0
- Now if 2 pages need to be replaced, which will be your victims?

Counting based algorithms

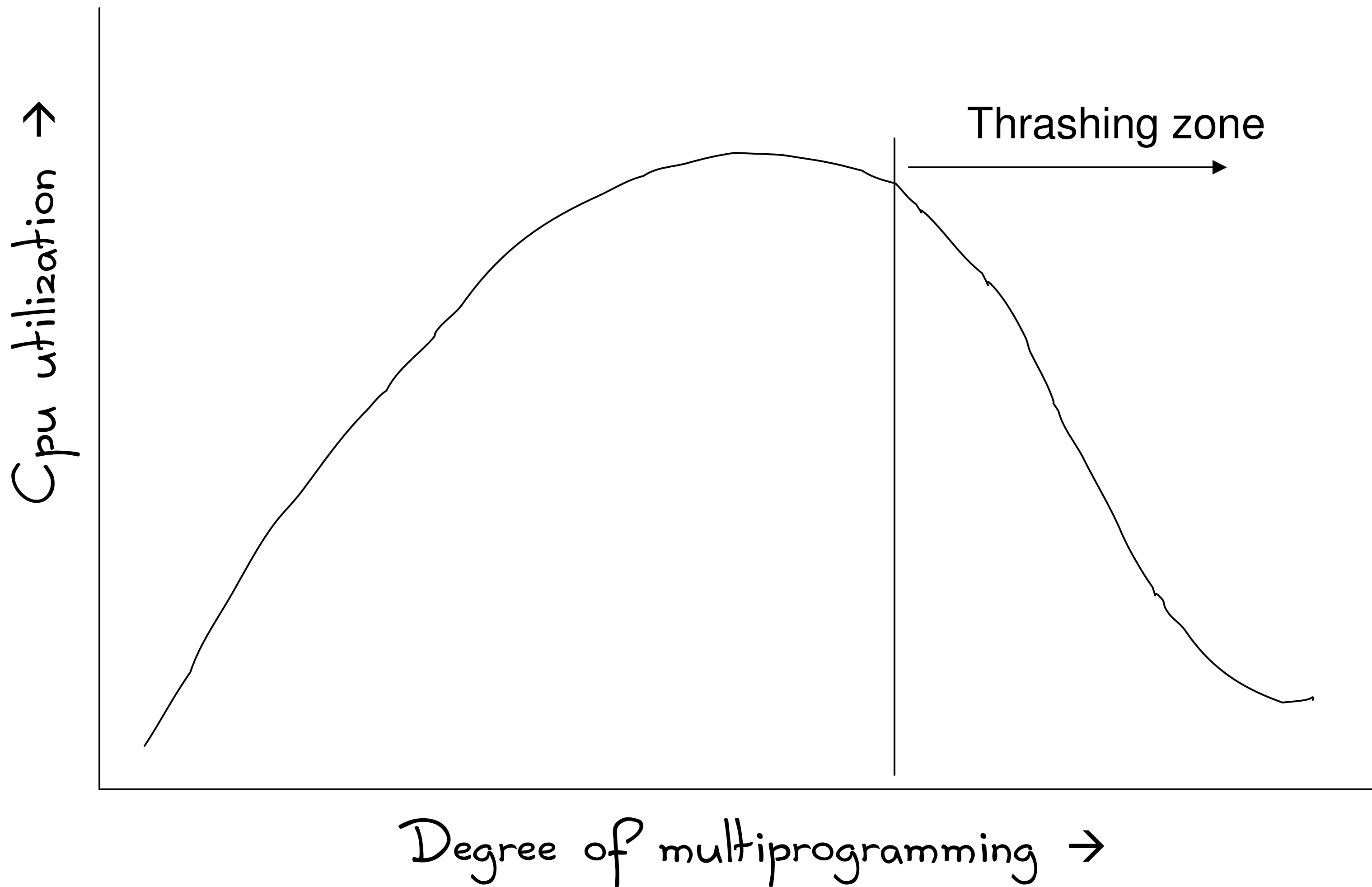
- Count of no. of references to each page (costly implementation)
  - Least frequently used
  - Most frequently used

Thrashing

- A process may have more than min pages required currently, but it page faults again and again since its active pages get replaced
- Early systems: OS detected less CPU utilization and introduced new batch processes → resulted in lesser CPU utilization!
- Effect of thrashing could be localized by using local page replacement as against global page replacement

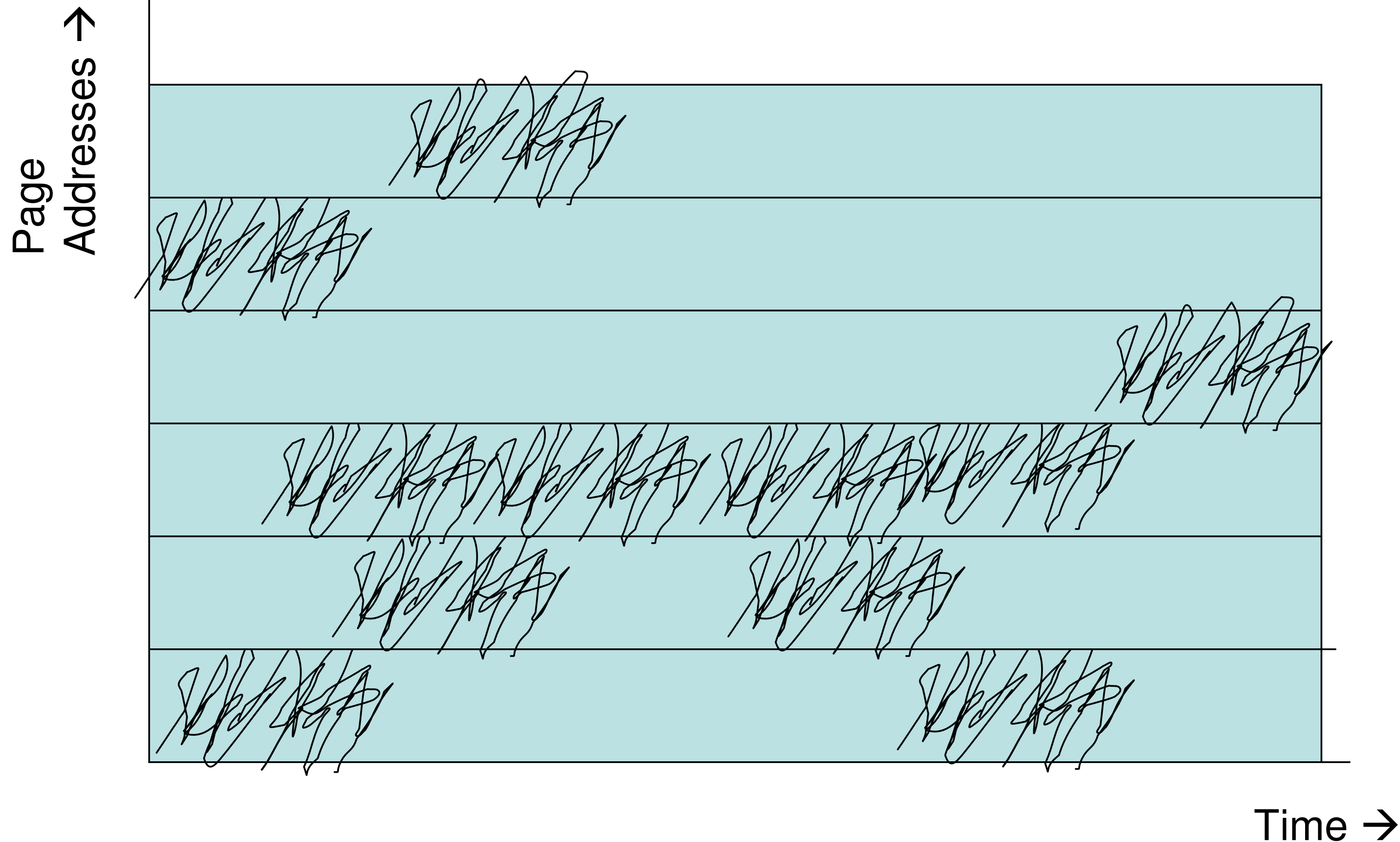
Thrashing

Thrashing process spends more time in paging than executing



Locality of Reference

Allocate as many frames to a process as in its current locality
(e.g. a function code and a global data structure)



The working set model for page replacement

- Define working set as set of pages in the most recent *d* references

- Working set is an approximation for locality

An example

- d=10
- Reference String:
- 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4

t1 →

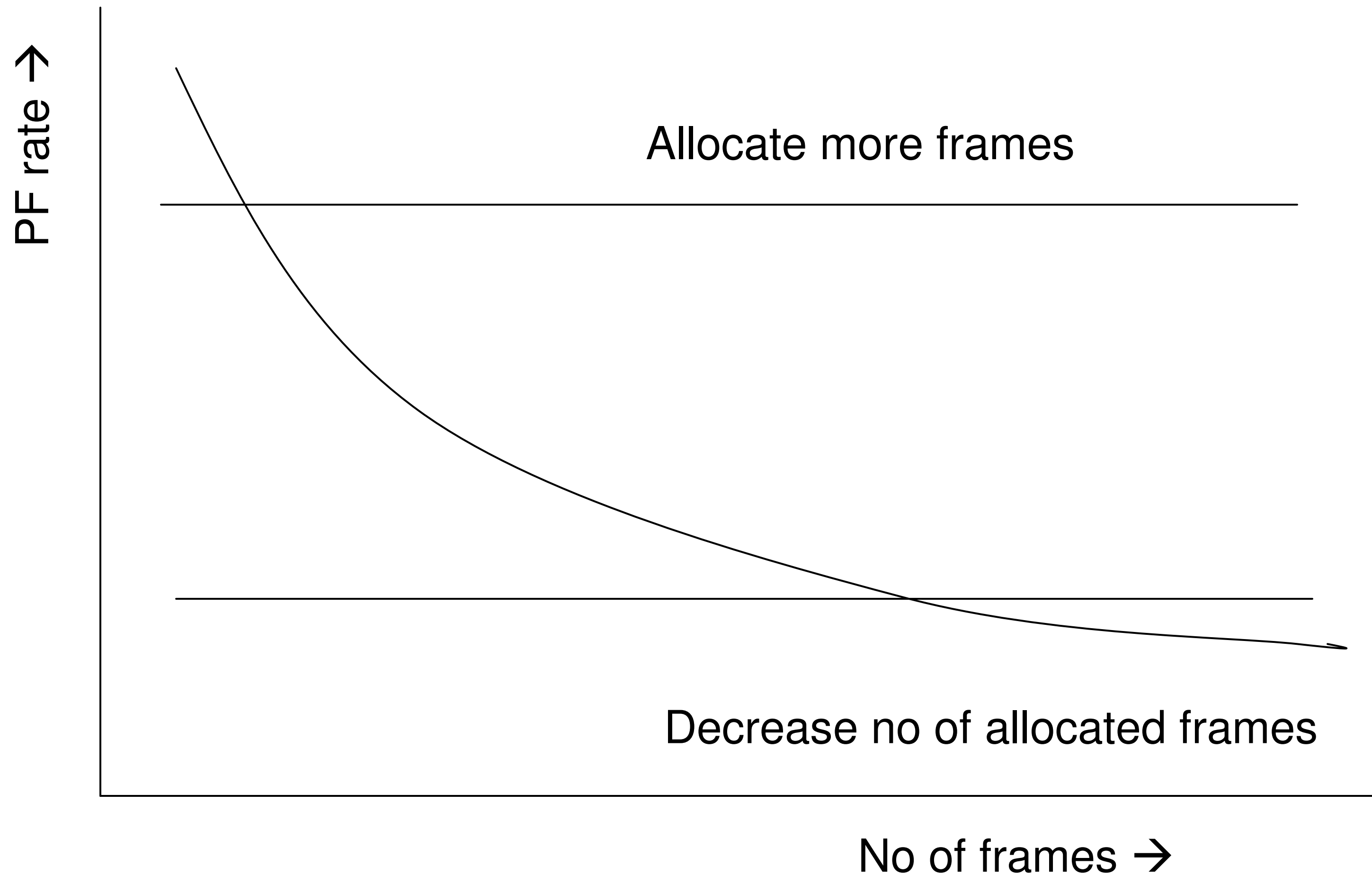
t2 →

- WS(t1) = {1,2, 5, 6, 7}
- WS(t2) = {3, 4}

What should be the value of d?

- If d is too small, it may not cover the current locality
- If too large, it overlaps several localities
- $D = \sum_{\forall i} \text{sizeof}(WS_i)$ = total demand for pages
- If $D > m$ (#frames): Thrashing will occur
- OS selects a process and suspends it (swap out)

Page fault rate for a process: for page replacement



Preallocation and allocation control

- Preallocation: start with a min no of frames per process
- Imposing limits: Administrator imposes upper bound on pages per process/per user

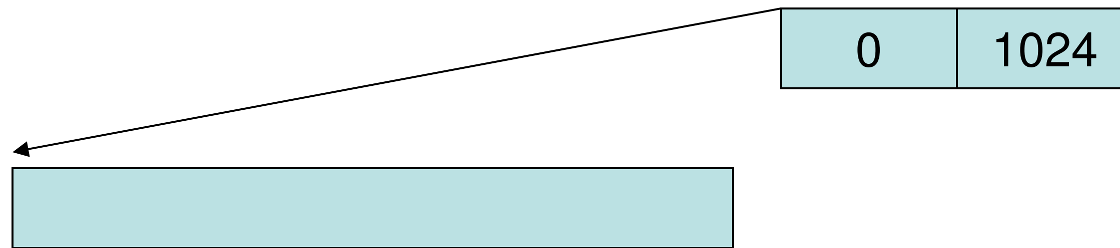
Kernel memory allocator

- Must handle both Small and large chunks
- Page level allocator is not suitable
- E.g. pathname translation, interrupt handlers, zombie exit status, proc structures, file descriptor blocks
- Page level allocator may preallocate pages to kernel, and kernel may efficiently use it with an allocator on top of a page(s)
- Kernel memory may be statically allocated or kernel may ask page allocator for more pages if needed
- KMA shouldn't suffer much from fragmentation and also should be fast

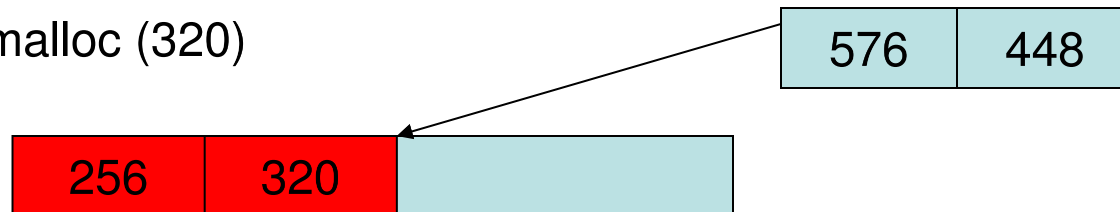
Resource Map Allocator

- Set of <base, size> pairs for free memory area

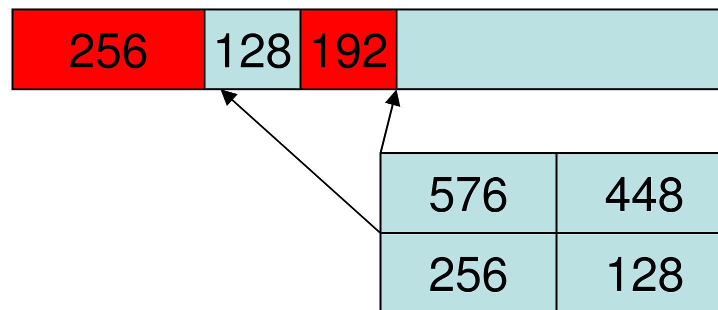
Initially:

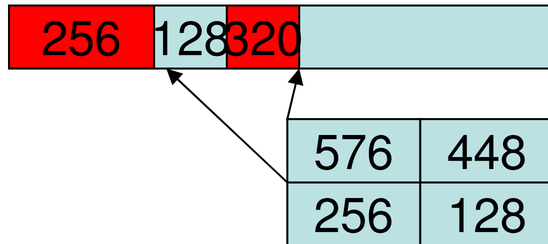


Call rmalloc (256), rmalloc (320)



Call rmfree (256,128)



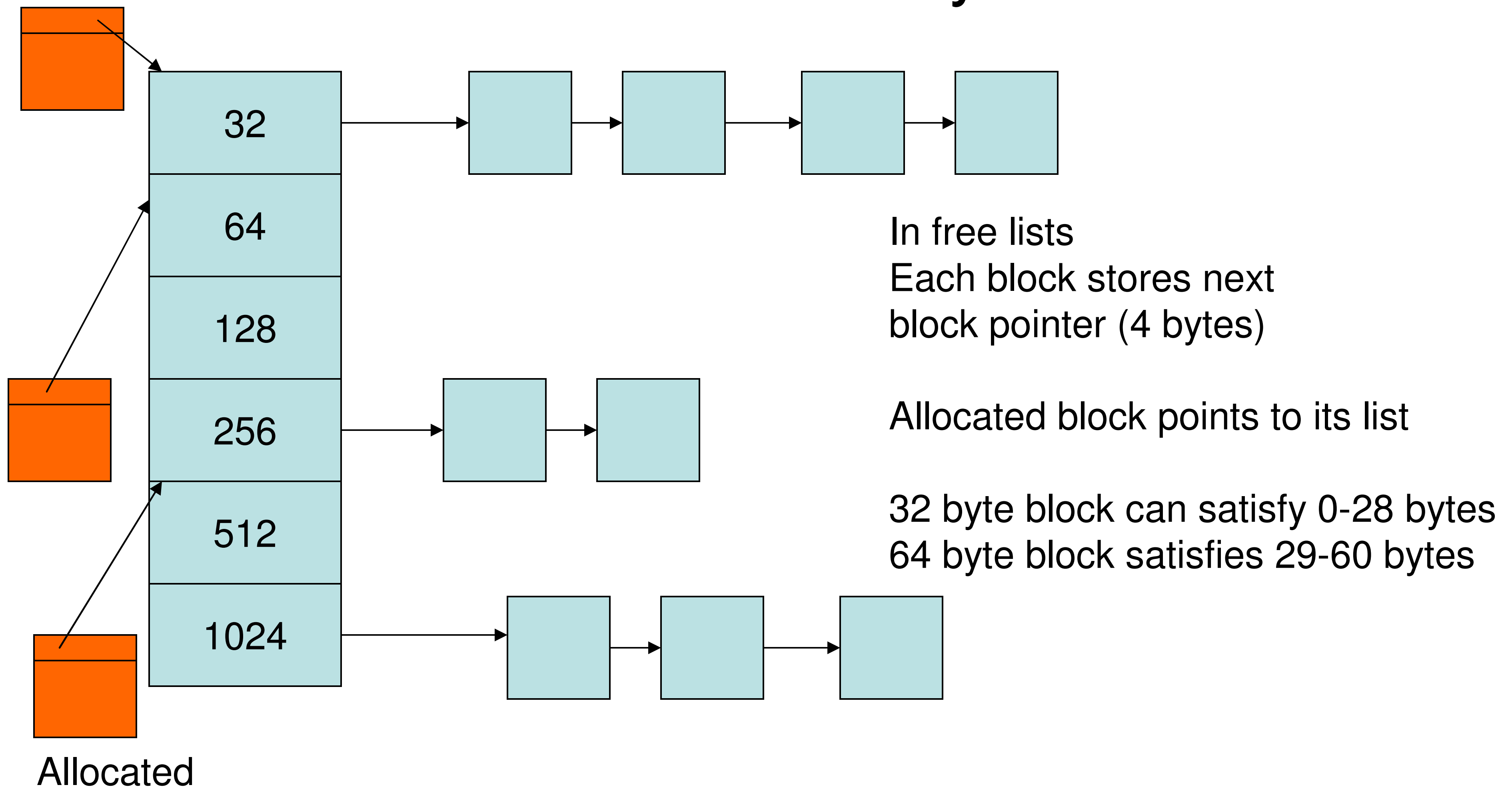


RMA

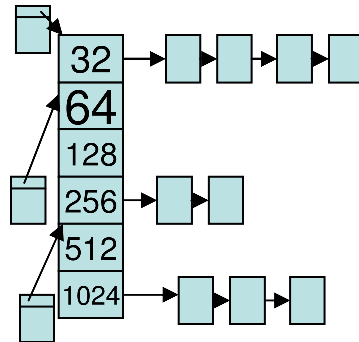
- After many memory operations, there may be many free blocks of varying sizes
- Unix uses typically first fit to find enough memory (linear search through list)
- As size of resource map increases fragmentation increases
- To coalesce adjacent regions, map must be kept in sorted order
- 4.3 BSD, System V IPC in some cases

Simple Power-of-Two free lists

- A list stores buffers of 2^k bytes



Simple Power-of-Two free lists

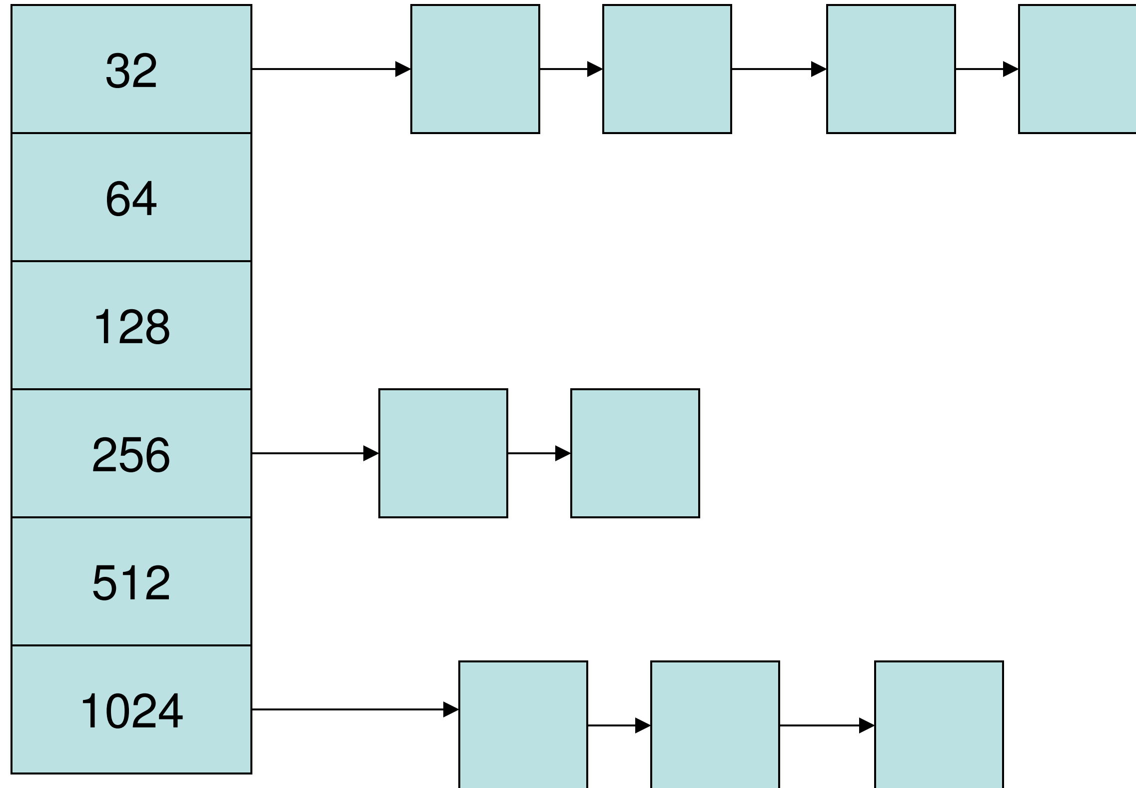


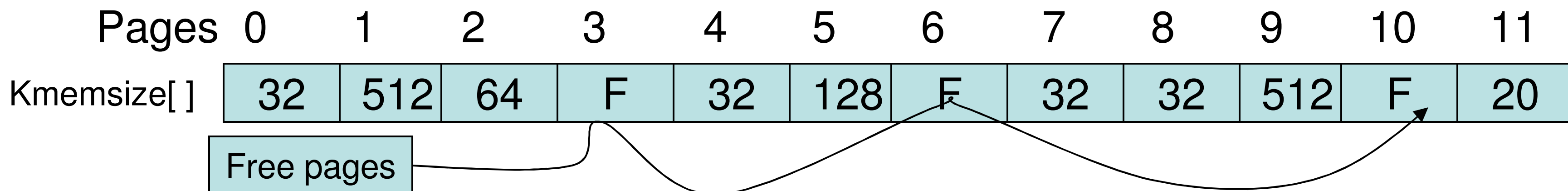
- Avoids lengthy (linear) search
- Internal fragmentation: e.g. for 512 bytes, use 1024
- No coalescing as sizes are fixed

Mc Kusick – Karels Allocator

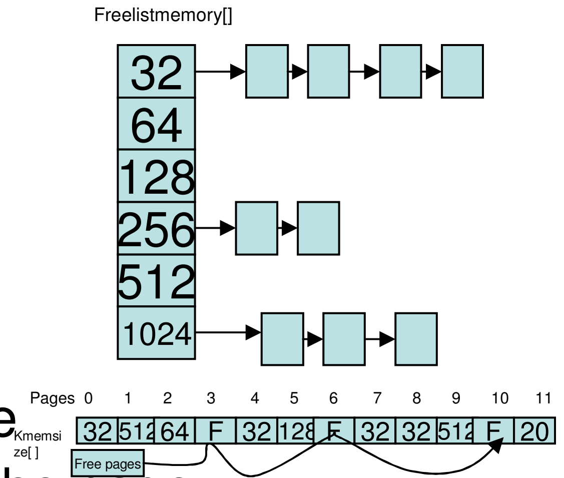
- Improvements over power-of-two allocator
- 4.4 BSD, DIGITAL Unix
- No wastage when requested memory is exactly power of two

Mc kusick- Karels allocator





MK Allocator

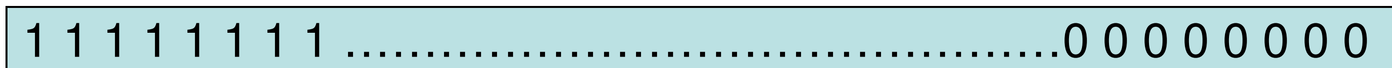


- Kmemsize[ ]
  - If the page is free, it indicates next free page
  - Else it contains block size used to partition the page
- Allocated blocks do not contain the address of free memory list to which they are to be returned
- We know to what page an allocated block belongs from MSB of the block (page no.)
- This page no. is used to find out size of the block through kmemsize [ ] array

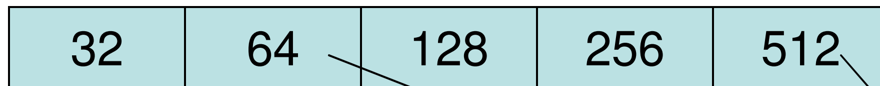
Buddy Allocator

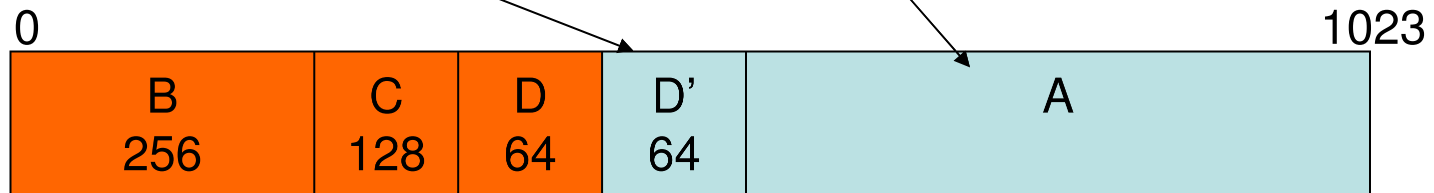
- Create small buffers by repeatedly halving a large buffer
- Coalescing of adjacent buffers whenever possible
- Each half is a ‘buddy’ of the other

Buddy Allocator

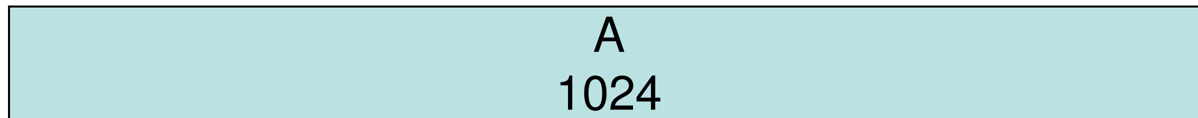


Free list headers:





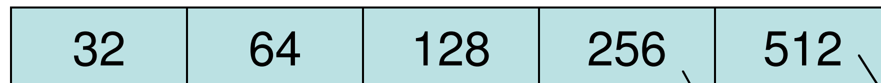
Buddy Allocator

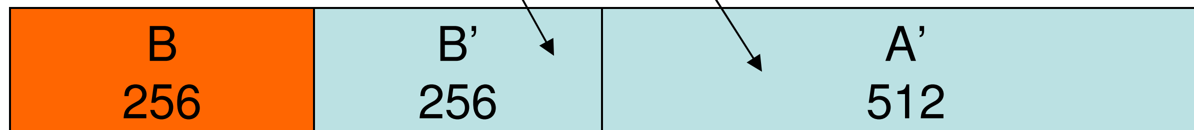


Now Allocate 256

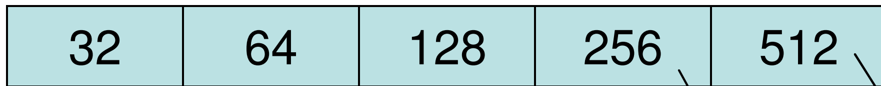
split A into A, A’
split A into B, B’
return B

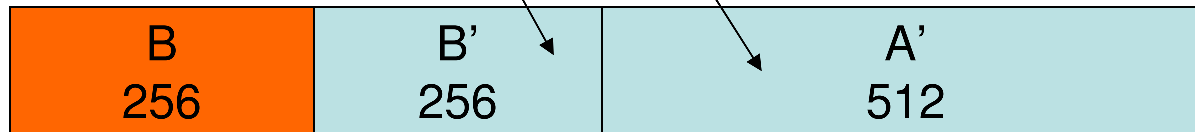
Free list headers





Free list headers



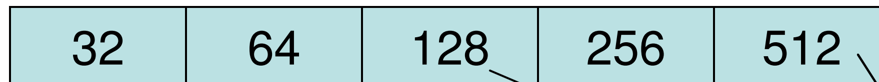


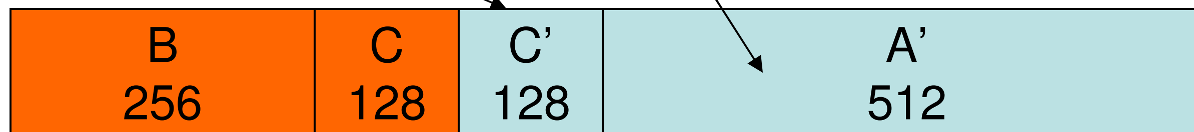
Now Allocate 128

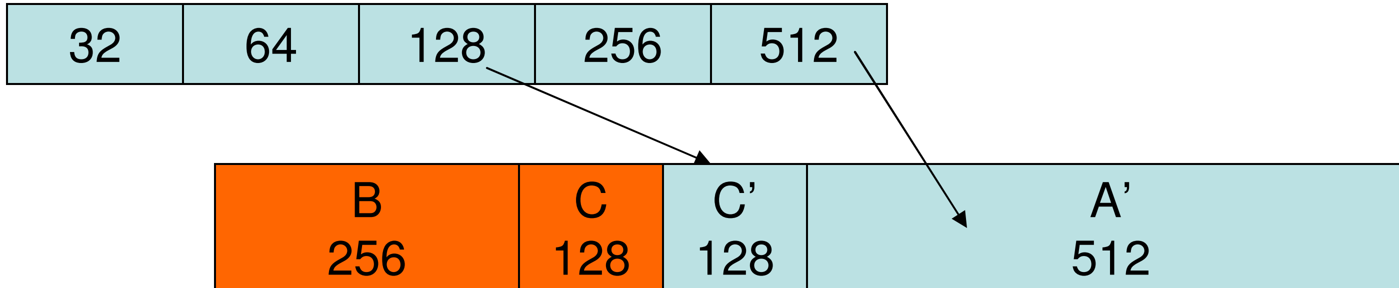
split B' into C and C'

allocate C

Free list headers



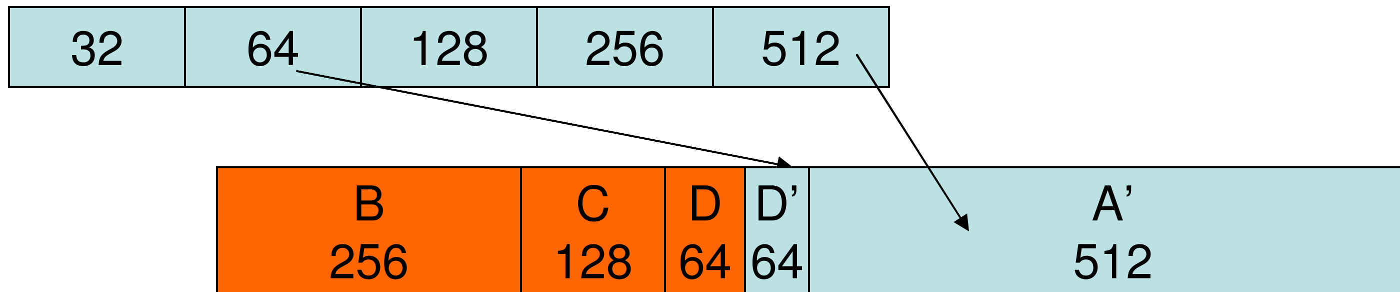




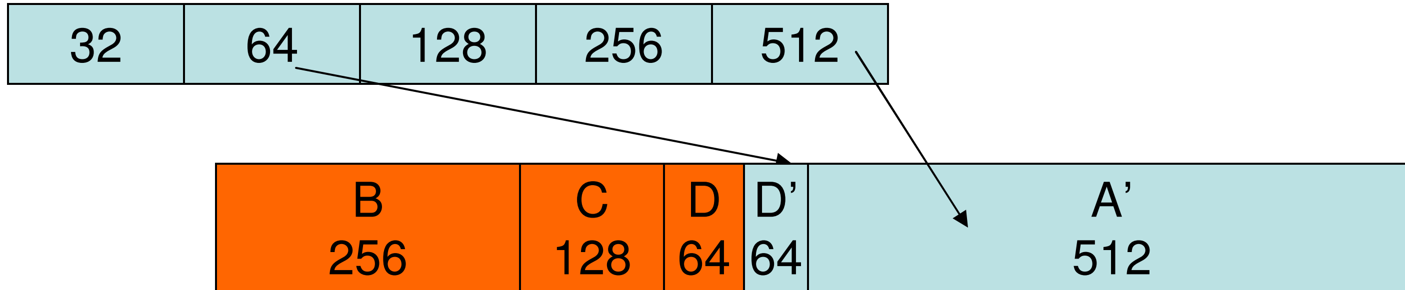
Now Allocate 64

split C’ into D and D’

allocate D



Free list headers



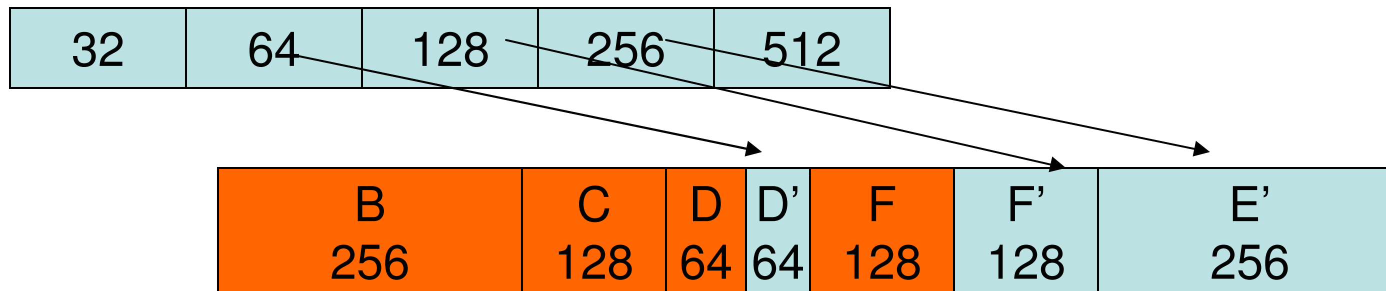
Now Allocate 128

split A’ into E and E’

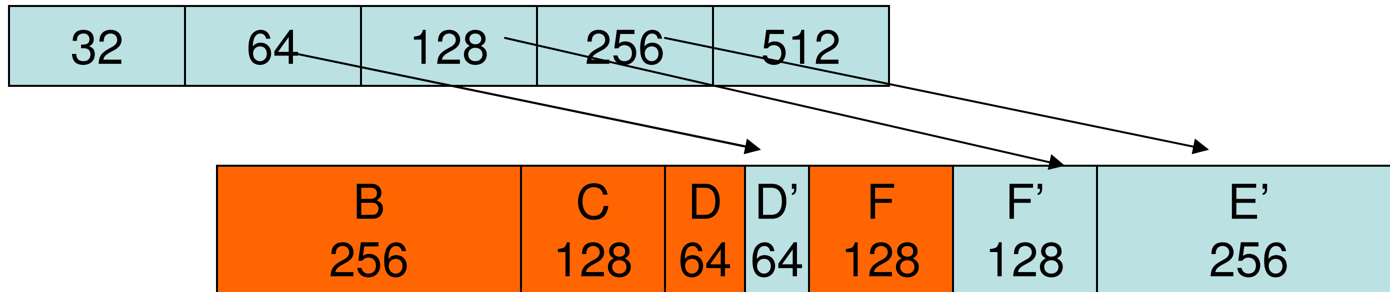
split E into F and F’

allocate F

Free list headers

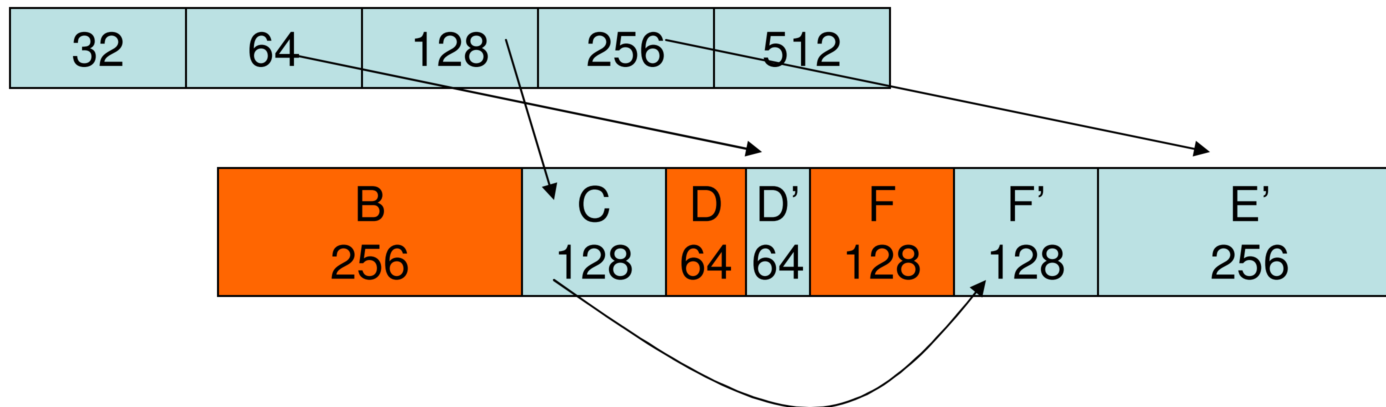


Free list headers

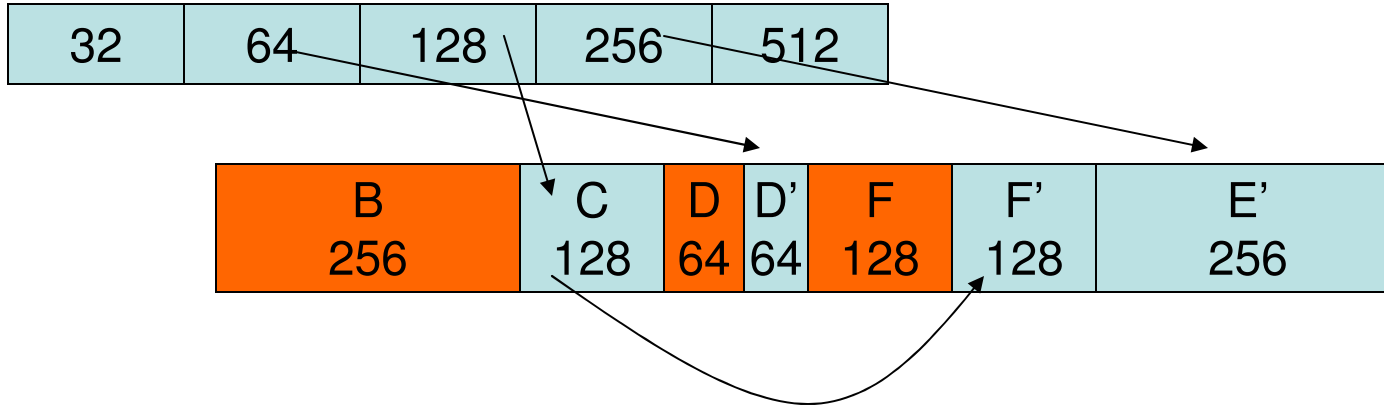


Now release (C, 128)

Free list headers



Free list headers



Now release (D, 64)

Free list headers