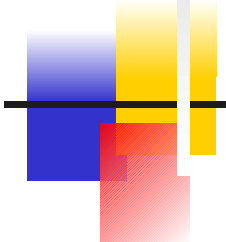


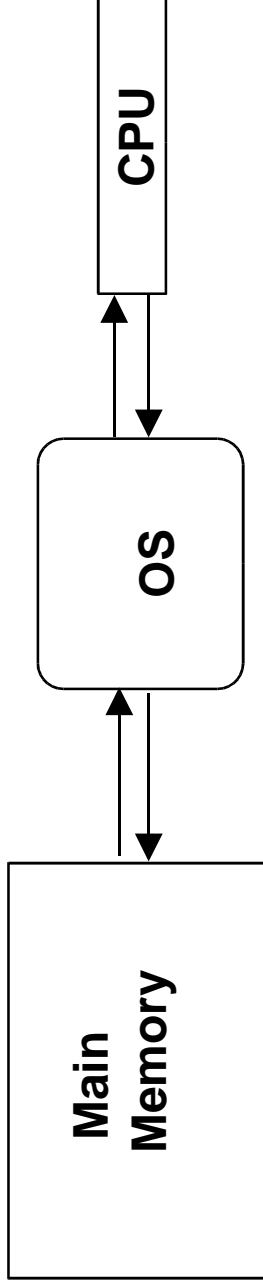
Chapter 11

Memory Management



Memory Management

Main memory is a resource that must be allocated and deallocated



Memory Management Techniques determine:

- Where and how a process resides in memory
- How addressing is performed

Binding:

identifiers --> compiled relative addresses (relative to 0)

--> physical addresses



Memory Management Techniques

- 1) Single Contiguous
- 2) Overlays
- 3) Fixed (Static) Partitions
- 4) Relocation (Dynamic) Partitions
- 5) Paging
- 6) Demand Paging
- 7) Segmented
- 8) Segmented / Demand Paging

For each technique, observe:

- Algorithms
- Advantages / Disadvantages
- Special Requirements



I. Single Contiguous

```
While ( job is ready ) Do  
  If ( JobSize <= MemorySize )  
    Then Begin  
      Allocate Memory  
      Load and Execute Job  
      Deallocate Memory  
    End  
  Else Error
```



I. Single Contiguous...

☺ Advantages:

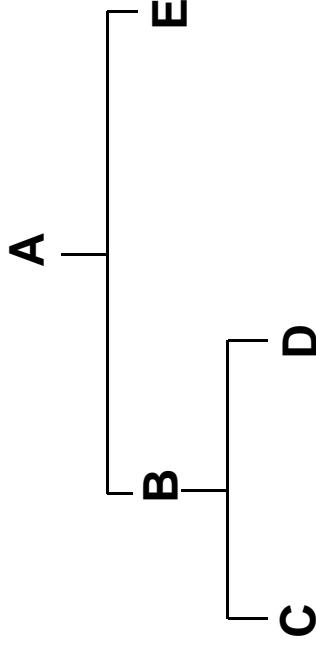
- Simplicity
- No special hardware

☹ Disadvantages:

- CPU wasted
- Main memory not fully used
- Limited job size

II. Overlays

- Programs can be sectioned into modules
- Not all modules need to be in main memory at the same time



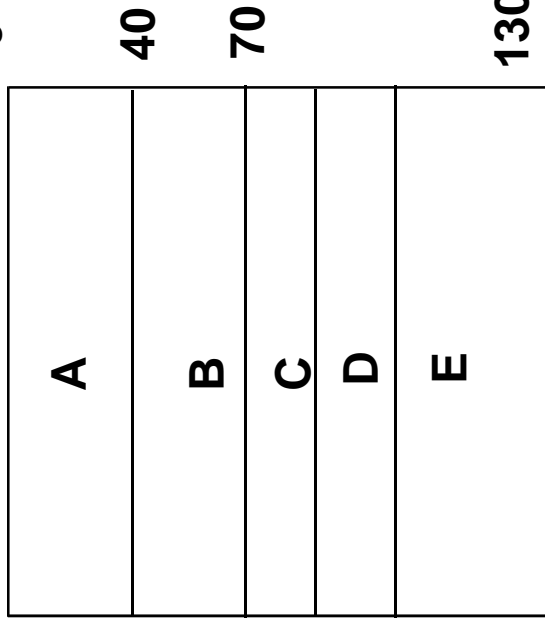
- Programmer specifies which modules can overlay each other
- Linker inserts commands to invoke the loader when the modules are referenced
- The "parent" must stay in memory
- Used in DOS as an alternative to Expanded Memory.

Illustration of Overlays

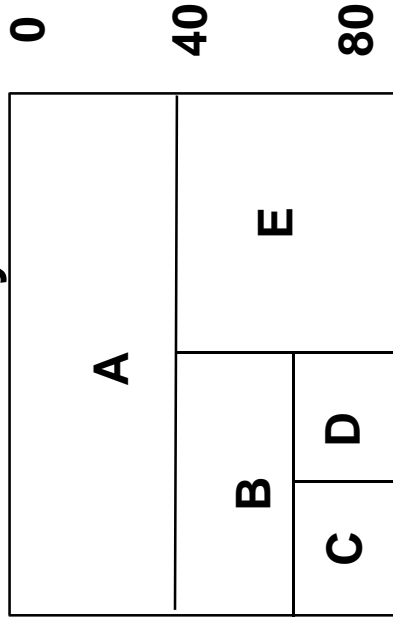
Program Component: A B C D E

Memory: 40K 30K 10K 10K 40K

Without
Overlays



With
Overlays





Overlays ...

☺ Advantages:

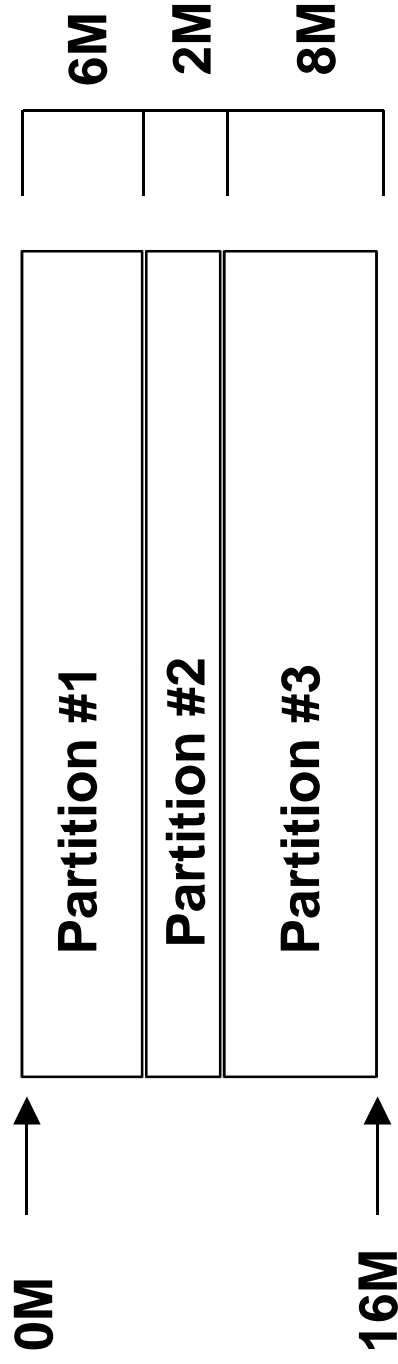
- Reduced memory requirements

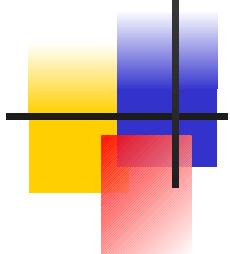
☹ Disadvantages:

- Overlap map must be specified by programmer
- Programmer must know memory requirements
- Overlapped modules must be completely disjoint

Fixed (Static) Partitioning with Absolute Translation

- Earliest attempt at multiprogramming
- Partition memory into fixed sized areas:





Fixed (Static) Partitioning with Absolute Translation ...

- Each partition can hold ONE process
- Code generated using an ABSOLUTE address reflecting the starting address of the partition in which it is supposed to execute (relative to 0, 6M, or 8M in picture)
- Queue of processes waiting for each partition

Fixed (Static) Partitioning with Absolute Translation

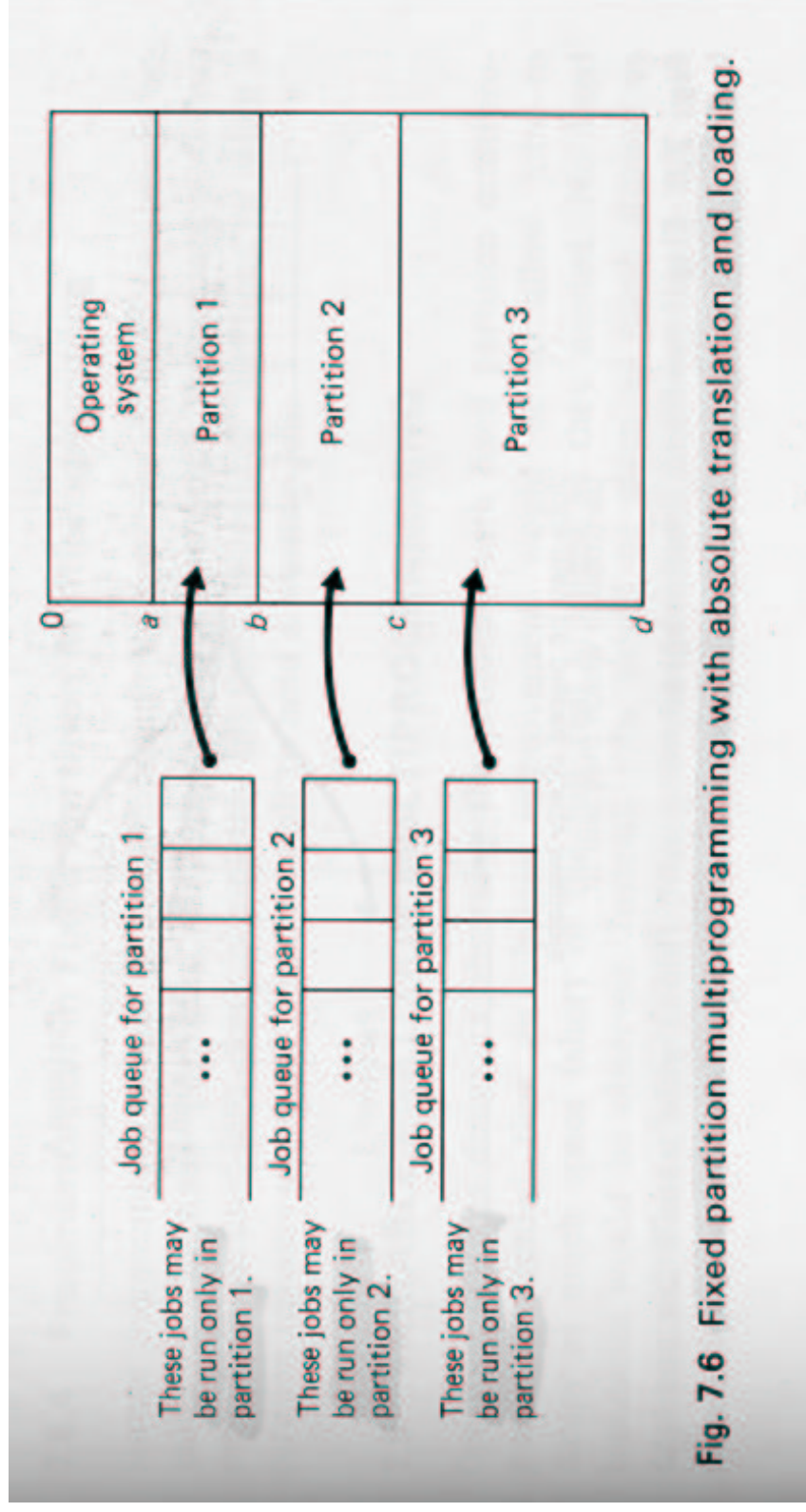


Fig. 7.6 Fixed partition multiprogramming with absolute translation and loading.

Fixed (Static) Partitioning with Absolute Translation...

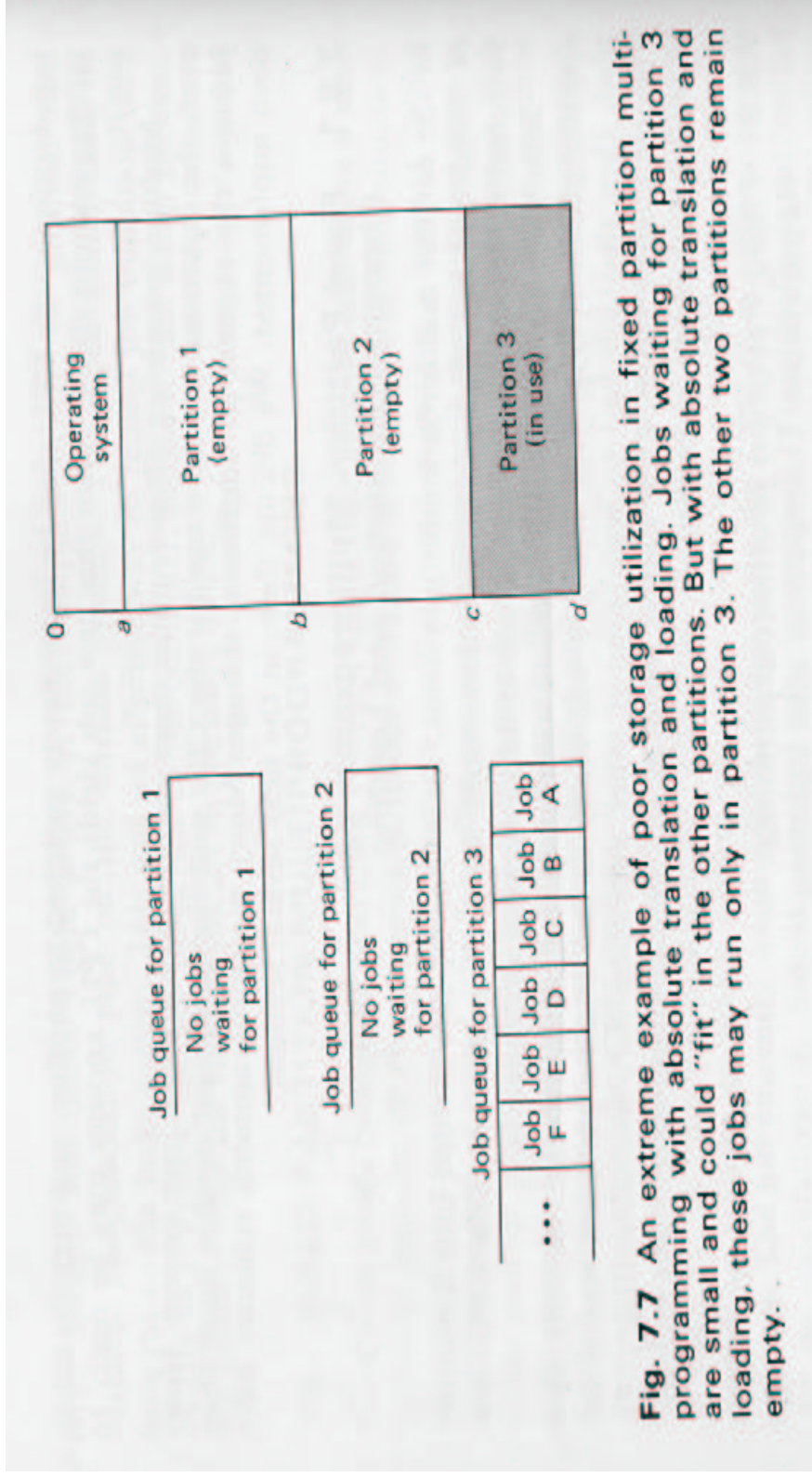


Fig. 7.7 An extreme example of poor storage utilization in fixed partition multi-programming with absolute translation and loading. Jobs waiting for partition 3 are small and could "fit" in the other partitions. But with absolute translation and loading, these jobs may run only in partition 3. The other two partitions remain empty.



Fragmentation– Definitions

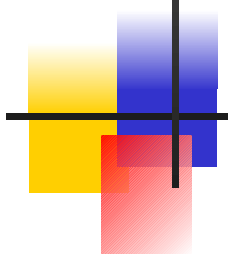
Fragmentation is a situation in which the free cells in main memory are not contiguous.

Internal fragmentation:

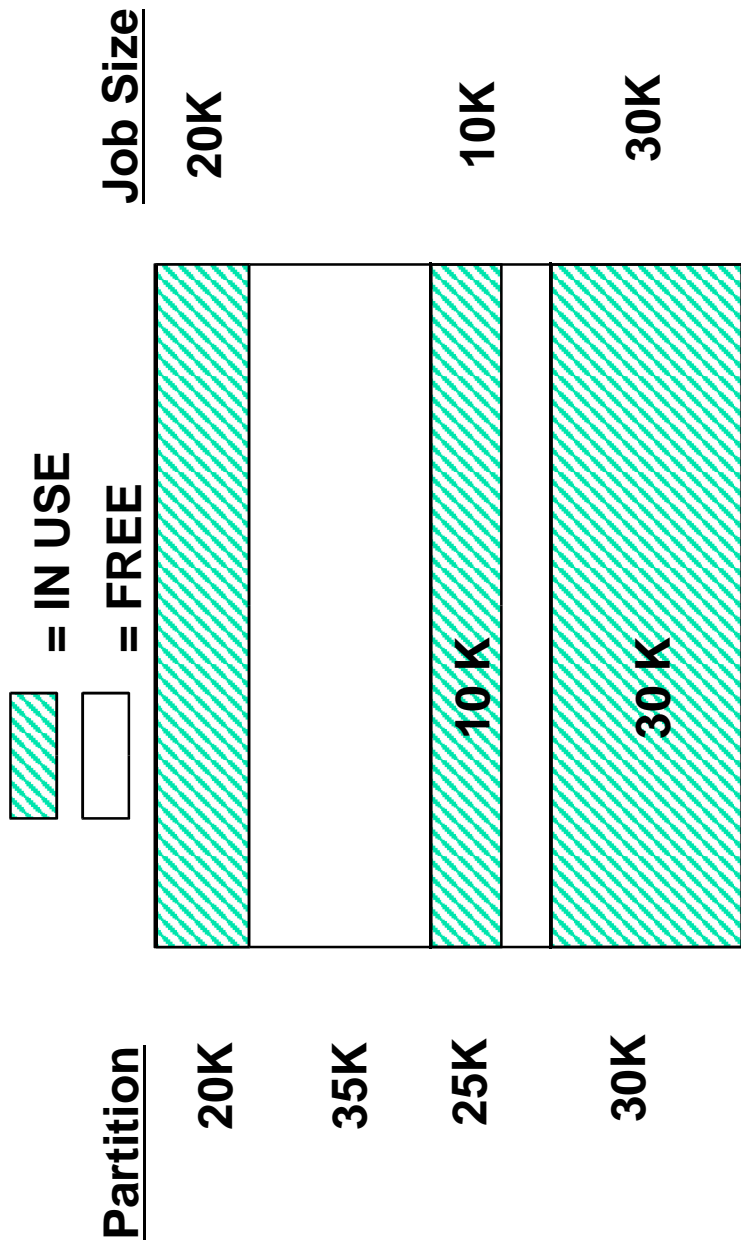
A situation in which free memory cells are within the area allocated to a process

External fragmentation:

A situation in which free memory cells are not in the area allocated to any process



Fixed Partition Fragmentation



External fragmentation: 35K partition

Internal fragmentation: 25-10 => 15K wasted inside 25K partition



Fixed Partitioning with Absolute Translation: Pros/Cons

☺ Advantages:

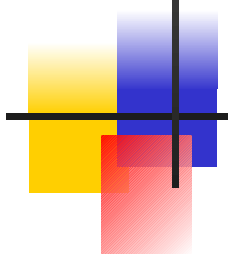
- Simplicity
- Multiprogramming now possible
- Works with *any* hardware (8088, 68000, etc)



Fixed Partitioning with Absolute Translation: Pros/Cons ...

☹ Disadvantages:

- Job Size \leq Max Partition Size \leq MM Size
- Storage wasted due to internal fragmentation:
process size $<$ partition size
- Storage wasted due to external fragmentation:
A partition may be idle because none of the jobs assigned to it
are being run
- Once compiled a job can *only* be executed in
designated partition



Fixed (Static) Partitions with Relative Address Translation

- **Allows process to run in any free partition**
- **ALL Code generated using addresses *relative to zero***

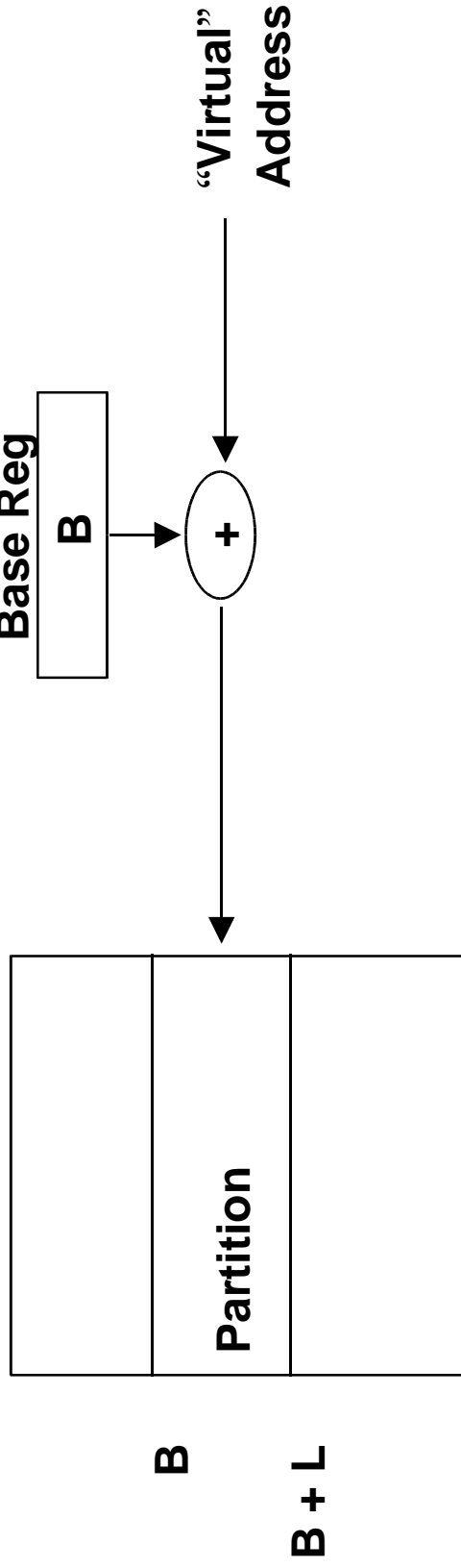
Fixed Partitions with Relative Address Translation...

Illustration:

Let:

B denote base (absolute) address of a partition

L denote partition length



QTP: Would Pointers work?



Multiprogramming Protection

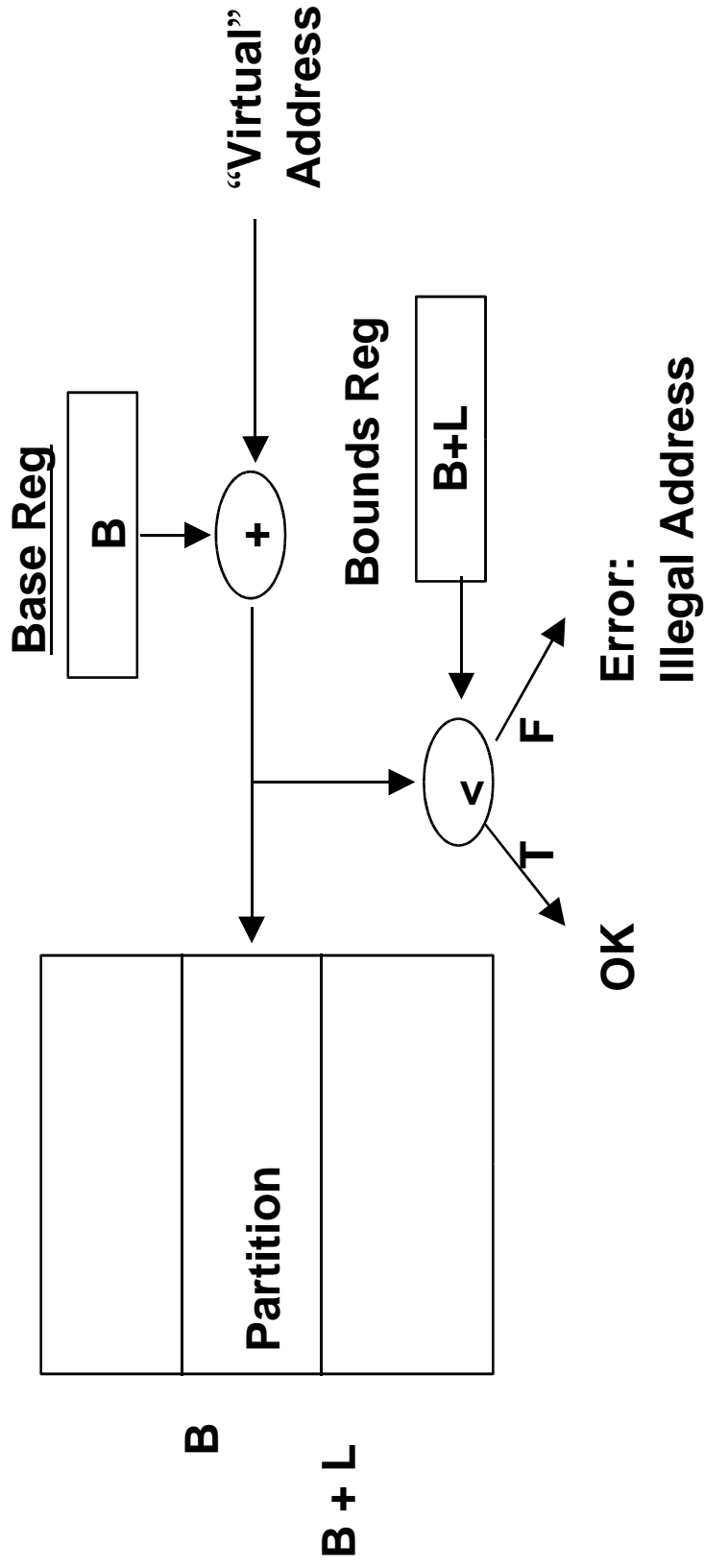
Fixed partitions with relative addressing supports multiprogramming protection

=> Ensure that one process does not access memory space dedicated to another process

Method:

Each relative address is compared to the **bounds register**

Multiprogramming Protection...





Fixed Partitioning with Relative Addressing: Pros/Cons

- ☺ Advantage compared to absolute addressing:
 - Dynamic allocation of programs to partitions improves system performance
- ☺ Still some disadvantages:
 - Partition sizes are fixed at boot time
 - Can't run process larger than largest partition
 - Partition selection algorithm affects system performance
 - Still has internal and external fragmentation

IV. Dynamic Partitions

Consider following scenario (100K memory):

1. Job 1 arrives; size= 22 K
2. Job 2 arrives; size= 24 K
3. Job 3 arrives; size= 30 K
4. Job 4 arrives; size=10 K
5. Job 1 terminates
6. Job 3 terminates
7. Job 5 arrives; size=12K

Where should job 5 be put?

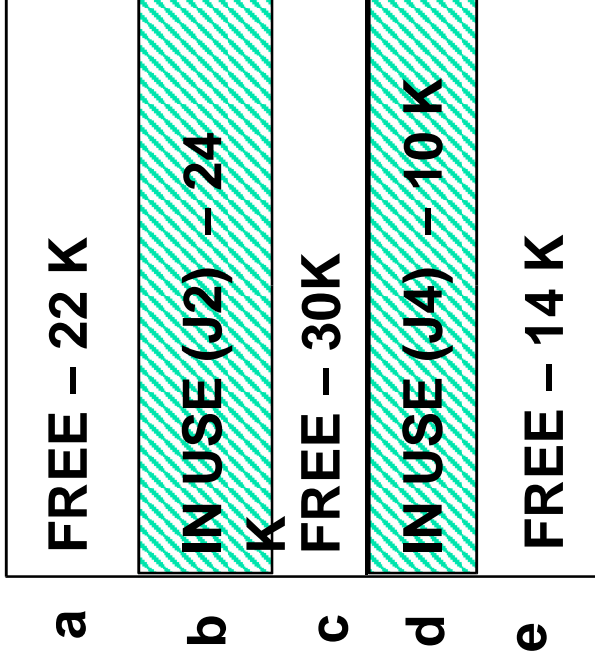




Partition Selection Algorithms

- Implementation requires a free block table
- Sorting table in a particular manner results in a specific selection algorithm:
 - 1) First Fit --- Table sorted by location, searched top to bottom
 - 2) Best Fit --- Table Sorted by size (ascending)
[don't break up big blocks]
 - 3) Worst Fit --- Table sort by size (descending)
[break up big blocks]
 - 4) Next Fit

Where does Job 5 Go? First Fit



⋮
⋮
7. Job 5 arrives; size=12K

Free List Table - First Fit

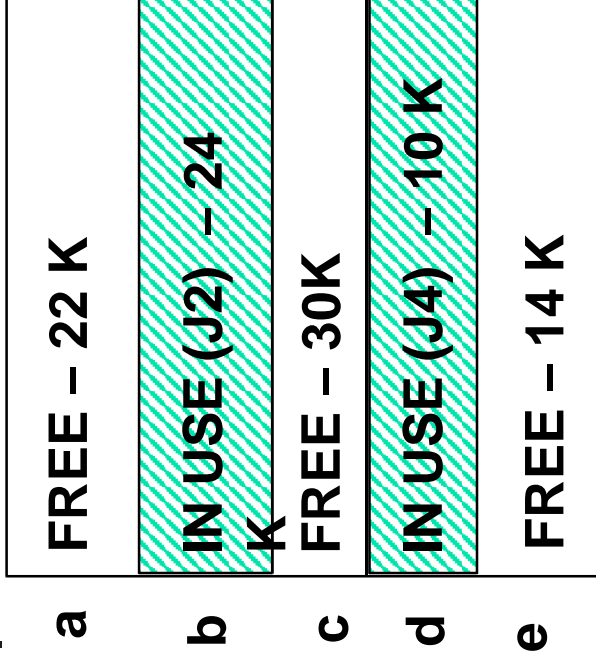
Start addr Length

a 22

c 30

e 14

Where does Job 5 Go? Best Fit



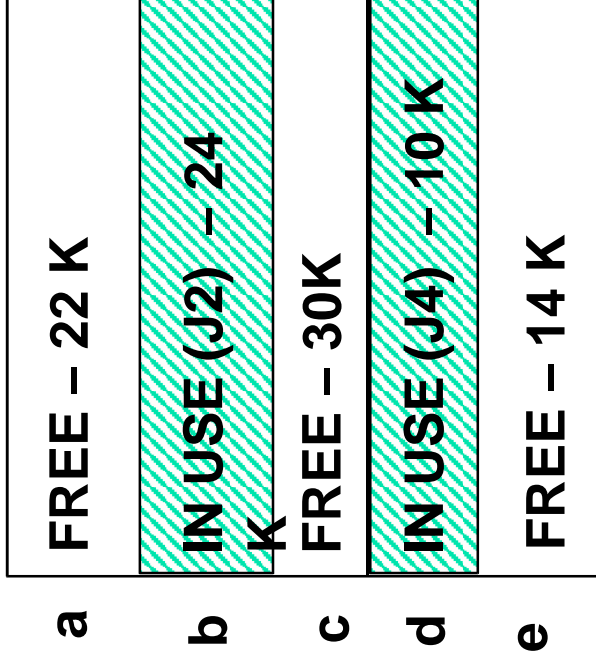
Free List Table - Best Fit

Start addr Length

e 14
a 22
c 30

:
:
7. Job 5 arrives; size=12K

Where does Job 5 Go? Worst Fit

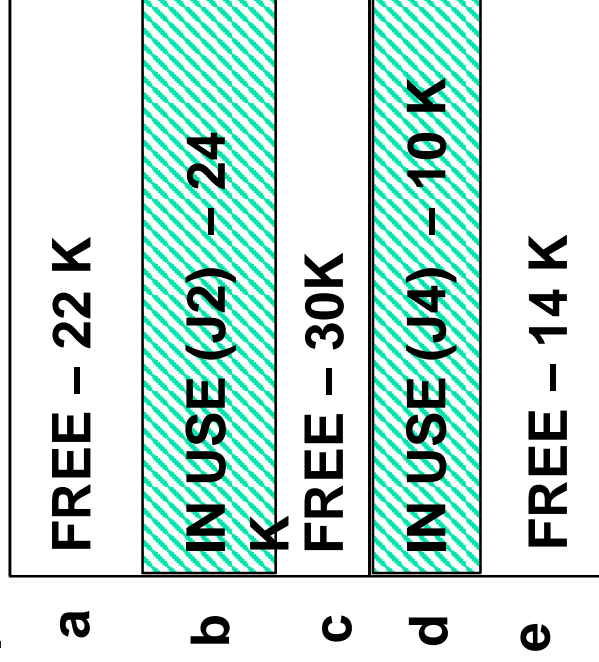


Free List Table – Worst Fit

<u>Start addr</u>	<u>Length</u>
c	30
a	22
e	14

:
:
7. Job 5 arrives; size=12K

Where does Job 5 Go? Next Fit



Free List Table – Next fit

<u>Start addr</u>	<u>Length</u>
a	22
c	30
e	14



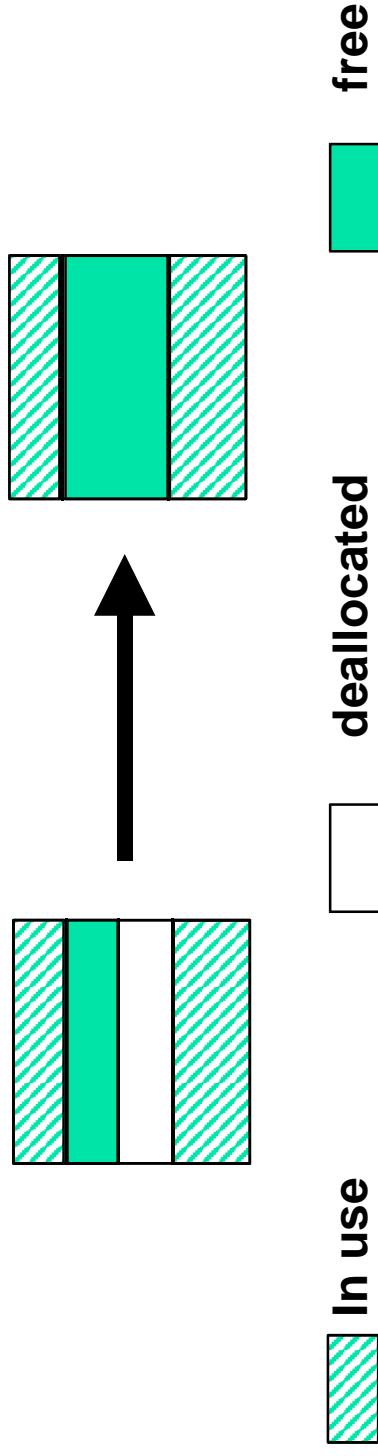
:
:
7. Job 5 arrives; size=12K



Dynamic Partitions

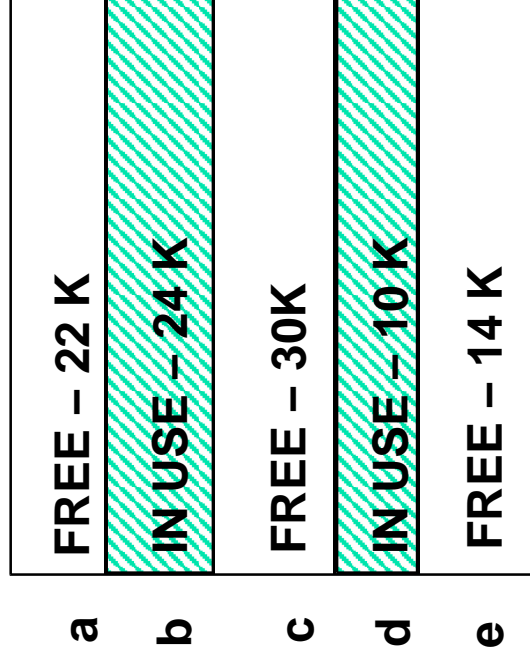
Requires two OS operations:

- Allocation:
Form a partition from a free partition of ample size
- Deallocation:
Return partition to free table and merge where possible



Merge Example

Suppose b becomes free



Free List Table - First Fit

Start addr Length

a 22

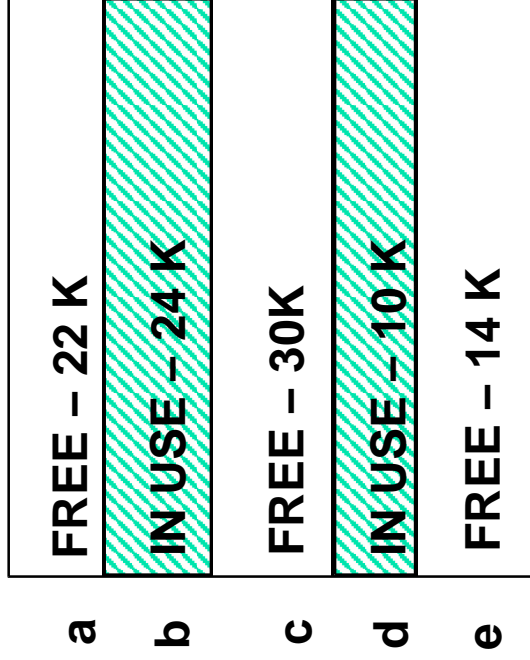
c 30

e 14

What does Free List Table look like?

Merge Example

Suppose b becomes free



Free List Table - Best Fit

Start addr Length

e 14

a 22

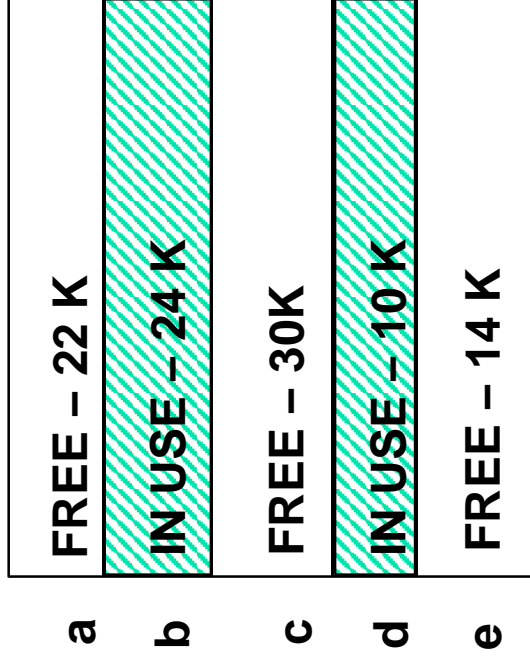
c 30

30

What does Free List Table look like?

Merge Example

Suppose b becomes free



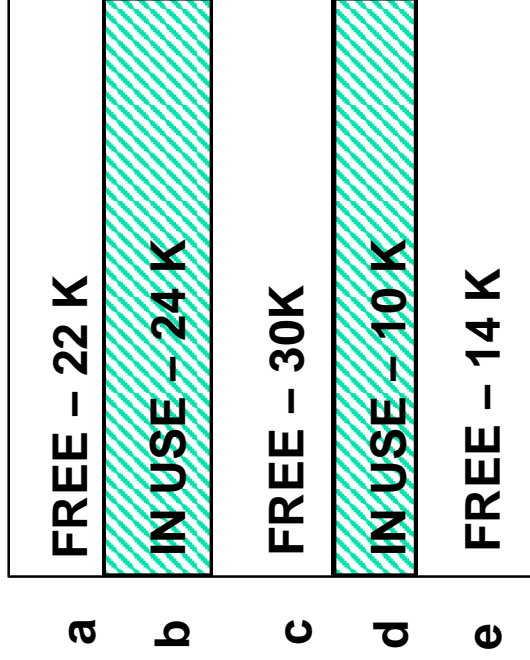
Free List Table – Worst Fit

	<u>Start addr</u>	<u>Length</u>
c		30
a		22
e		14

What does Free List Table look like?

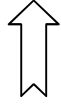
Merge Example

Suppose b becomes free



Free List Table – Next fit

	<u>Start addr</u>	<u>Length</u>
a		22
c		30
e		14



What does Free List Table look like?



What if we cannot find a big enough hole for an arriving job?

Suppose a 35K job arrives?

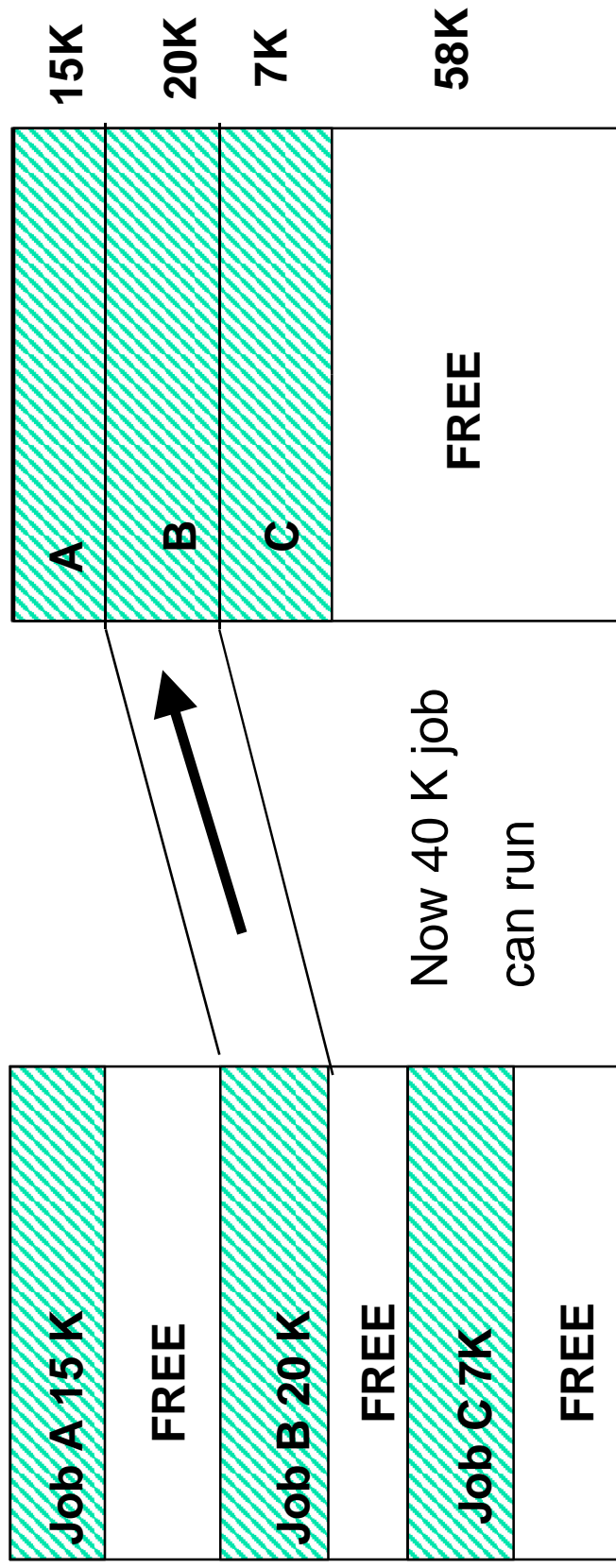
Suppose a 90K job arrives?

What do you do?

Free	22 K
2	24 K
Free	30 K
4	10 K
Free	14 K

Compaction

Shuffle jobs to create larger contiguous free memory



QTP: How about pointers?



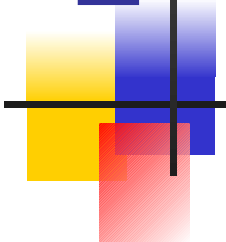
Pros/Cons of Dynamic Partitions

☺ Advantages:

- Efficient memory usage

☹ Disadvantages:

- Partition Management
- Compaction or external fragmentation
- Internal fragmentation (if blocks composing partitions are always allocated in fixed sized units --- e.g. 2k)



Multiple Segment Relocation registers

