

In multiprogramming CPU Utilizations dependence on the number of Process in MM

**2 CPU UTILIZATION ESTIMATE MM SIZE and IO USE EFFECTS.**

**3 CPU UTILIZATION DEPENDENCE**

**4 CPU UTILIZATION USE OF  $1-p^n / n$  TO COMPUTE TIME GIVEN TO SETS OF PROCESSES (SIMPLE EXAMPLE)**

**5 CPU UTILIZATION USE OF  $1-p^n / n$  TO COMPUTE TIME GIVEN TO SETS OF PROCESSES EXAMPLE (BOOK)**

Overview: Process-Hole MM assignment and Paging MM Assignment

**6 PROCESS ASSIGNMENT OF MEMORY-OVERVIEW**

So that only Process X has access to Process X's MM even though Process X may move

**7 RELOCATION AND PROTECTION HARDWARE**

Effect of process removal on generation of Holes. Schemes for finding a Hole to put a Process in.

**8 PROCESSES REMOVAL RESULTANT HOLES**

**9 PROCESSES AND HOLES INSERTION SCHEMES: FIRST, NEXT AND BEST FIT**

Keeping track of locations of Processes and Holes

**10 PROCESSES AND HOLES BIT MAP DATA STRUCTURE \*BLOCK SIZE Analysis\***

**11 PROCESSES AND HOLES LINK LIST DATA STRUCTURE**

**12 PROCESSES AND HOLES BINARY TREE DATA STRUCTURE BUDDY ALGORITHM**

More details on Buddy Algorithm

**13 BUDDY ALGORITHM MARKING TREE NODES TRACES**

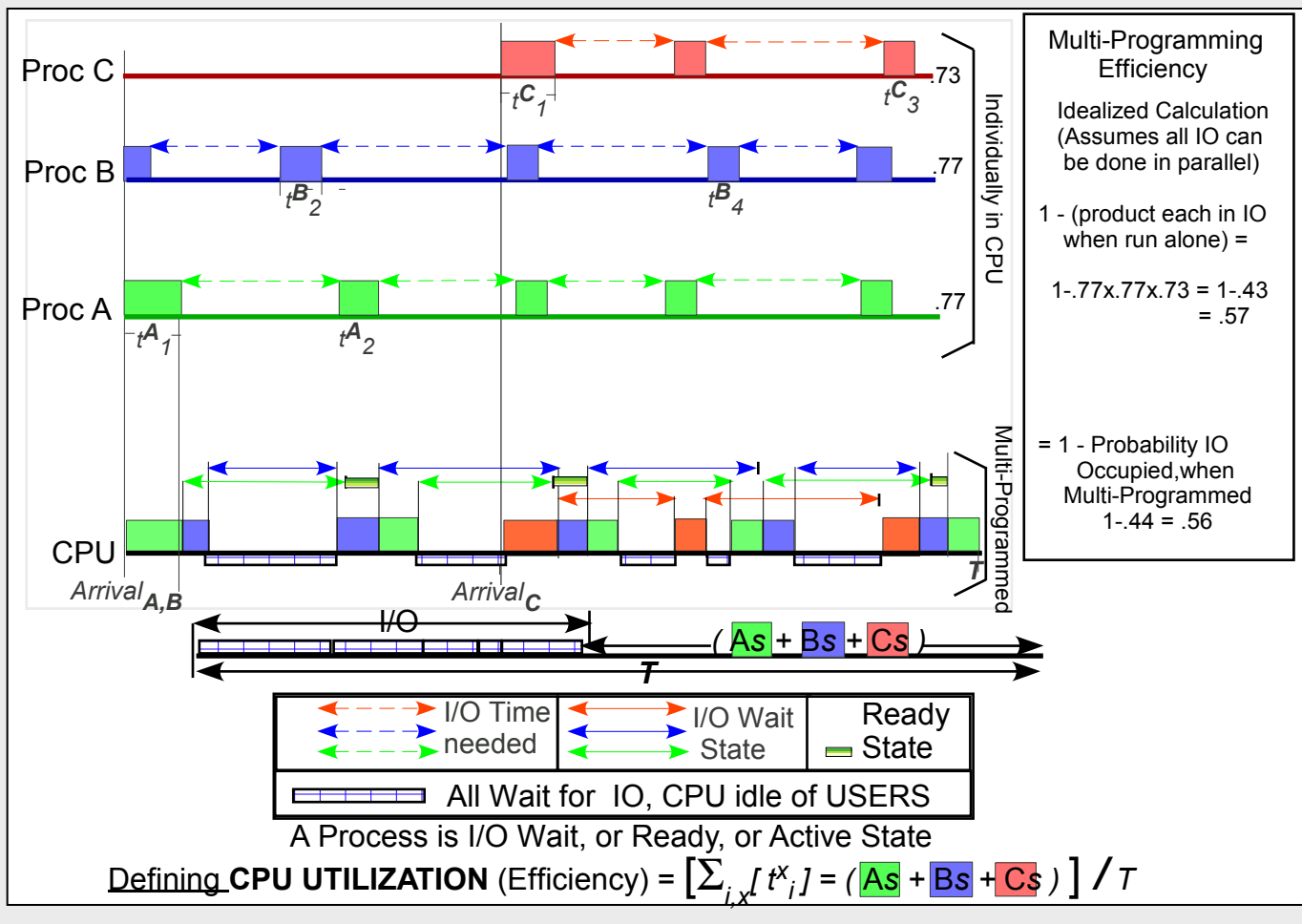
**14 PROCESSES AND HOLES Buddy Algorithm Example**

There are 1/2 as many holes as Processes (on average). Why? Consequence on assigning algorithms

**15 PROCESSES AND HOLES-THE 1/2 RULE**

**16 PROCESSES AND HOLES UTILIZATION ANALYSIS USING THE 1/2 RULE**

## CONTENTS



$n$  == Number of Processes (**degree of multiprogramming**) that are in MM The larger the Memory the larger  $n$  can be. For User Processes In MM:

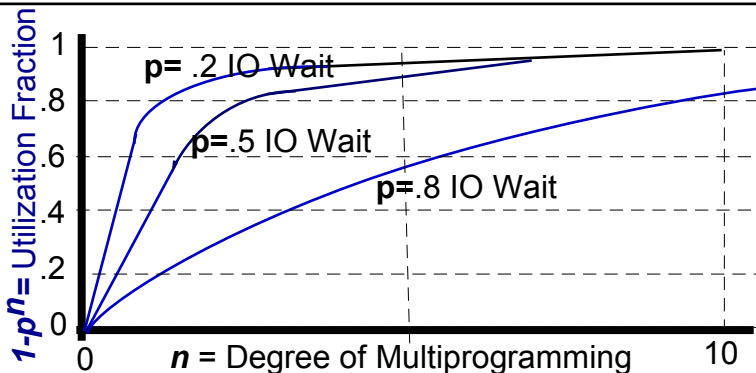
$p$  = (average) fraction of the time a User Processes **Waits for I/O\*** (running in Multi-Programming mode) [= Probability it is waiting for I/O]

$p^n$  = (average) fraction of the time all  $n$  User Processes are Waiting for I/O [= Probability all  $n$  are Waiting]

$1 - p^n$  = (average) fraction of the time a single User Process is in CPU. [=Probability all are Waiting]

\*Time Waiting for I/O to Completed >= Time in I/O. Time Waiting for I/O to Completed = Time in I/O iff all I/O requests can be handled in parallel

**Estimating CPU UTILIZATION**



Utilization is affected by many scheduling factors, such as the size of Quanta, the change in Priority associated with IO. In addition it is affected by the size of memory and the mix of Processes-and the percentage the time each requires IO.

**CPU UTILIZATION ESTIMATE MM SIZE and IO USE EFFECTS.**

Fraction Of Time	#Processes	1	2	3	4
	States				
CPU idle- All In I/O		$p$	$p^2$	$p^3$	$p^4$
CPU busy Not All in I/O		$1-p$	$1-p^2$	$1-p^3$	$1-p^4$
CPU per Process		$\frac{1-p}{1}$	$\frac{1-p^2}{2}$	$\frac{1-p^3}{3}$	$\frac{1-p^4}{4}$

If  $p = 3/4$ ,  $n = 2$  prob that Process in I/O

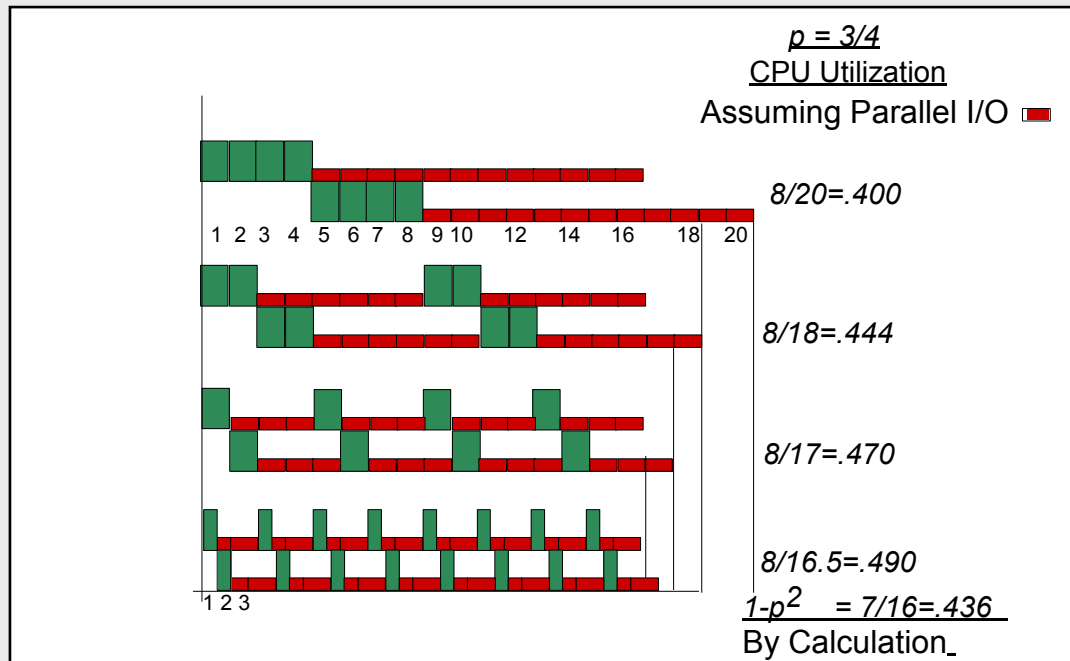
$p^n = 9/16$  prob that CPU idle-All in I/O

$1-p^n = 7/16$ , prob that CPU busy-not all in I/O

$[1-p^n] / 2 = 7/32$  fraction of CPU given to each Process

Table Of Ave Fraction Of CPU Time Available/Process As A Function Of Ave Fraction Of I/O Wait ( $p$ ) and Of Number Of Processes in MM ( $n$ )

Note: Assumes that if any are not in IO one is in CPU Of course those not blocked waiting for IO they may be blocked for other reasons. So the estimate of Utilization is likely to be lower than estimated this way.



Note: Though the Utilization seems to increase as the quanta size decreases we have not shown the cost of context switches. The number of context switches to complete a job increase as the quanta decrease. This will cause a decrease in Utilization and also increasing the Turnaround time for each job.

### How Actual Sequence of CPU and I/O Wait thus Quanta effects Actual Utilization

## CPU UTILIZATION DEPENDENCE

**Fraction of CPU for each Proc =  $1-p^n/n$**   
**BASIC  $T_{In CPU for eachProc} = T_{Run} \times 1-p^n/n$**   
 **$T_{Run} = T_{In CPU for eachProc} \times n / 1-p^n$**

**GIVEN:**

Jobs	Time Of Arrival	CPU is Needed
J1	0	10
J2	10	6

and Probability in I/O =  $p = .5$

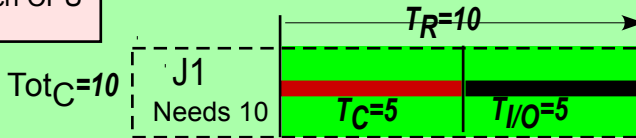
**PROBLEM: HOW LONG BEFORE J1 GETS ALL ITS CPU TIME?**

**Only First Job Present**

J1 runs alone for 10 time units  
How much CPU time?

1 JOB, J1, RUNNING

Run Time: 10



In run time  $T_R=10$  with a only J1 running

J1 gets  $T_C = T_R \times [(1-p)/1] = 10 \times .5 = 5$  in CPU (and  $T_{I/O}=5$  in I/O)

# of Jobs	Probability of:	
1		
All in IO (CPU Idle)	$p^1$	.5
CPU Busy	$1-p^1$	.5
$F_{C/P}$	$1-p^1/1$	.5

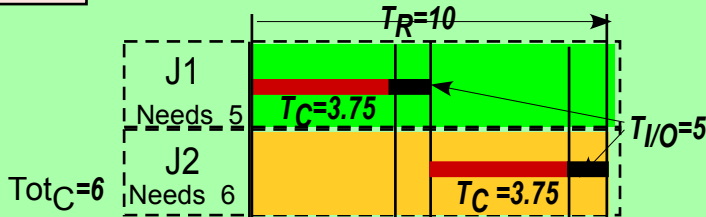
**Second Job Arrives**

J2 joins J1 for 10 time units.  
How much total CPU time?

CONTINUE

2 JOBS, J1 AND J2, RUNNING

Run Time: 10



In run time  $T_R=10$  with J1 and J2 both running

J1 and J2 each get  $T_C = T_R \times [(1-p^2)/2] = 10 \times .375 = 3.75$  in CPU (and  $T_{I/O}=1.25$  in I/O)

# of Jobs	Probability of:	
2		
All in IO (CPU Idle)	$p^2$	.25
CPU Busy	$1-p^2$	.75
$F_{C/P}$ CPU/Process	$1-p^2/2$	.375

CONTINUE

J1 needs a total of 10 time units in CPU to complete CPU use. How long till J1 completes?

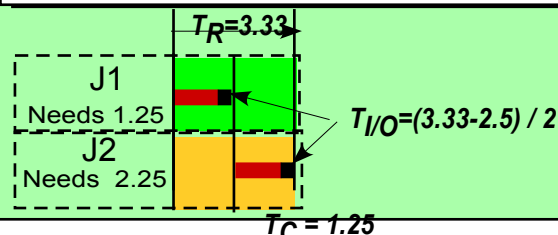
Now J1 has  $5 + 3.75 = 8.75$  needs  $1.25 =$  to complete its CPU time, ( $Tot_C=10$ )

Need: 1.25 CPU. Run Time = ?

$$T_C = T_R \times F_{C/P} (= [(1-p^2)/2])$$

$$T_C / F_{C/P} = T_R$$

$$1.25 / .375 = T_R = 3.33$$



**CPU UTILIZATION USE OF  $1-p^n/n$  TO COMPUTE TIME GIVEN TO SETS OF PROCESSES(SIMPLE EXAMPLE)**

# of processes $n$ probability of:	1	2	3
---------------------------------------	---	---	---

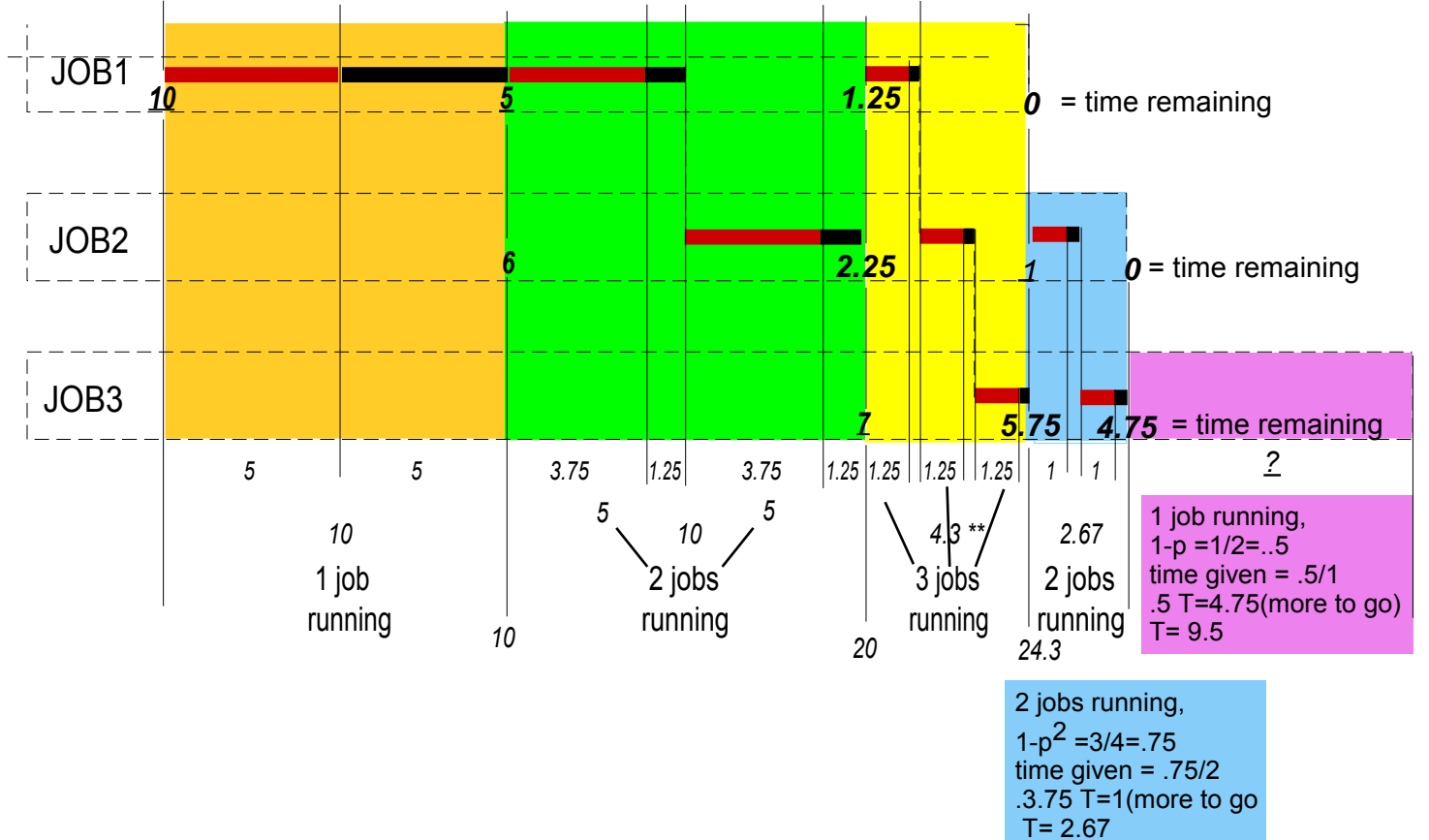
All in IO (CPU Idle)	$p^n$	.5	.25	.125
CPU Busy	$1-p^n$	.5	.75	.875
CPU/Process	$\frac{1-p^n}{n}$	.5	.375	.291

times jobs	arrival	duration in CPU
---------------	---------	--------------------

1	0	10
2	10	6
3	20	7

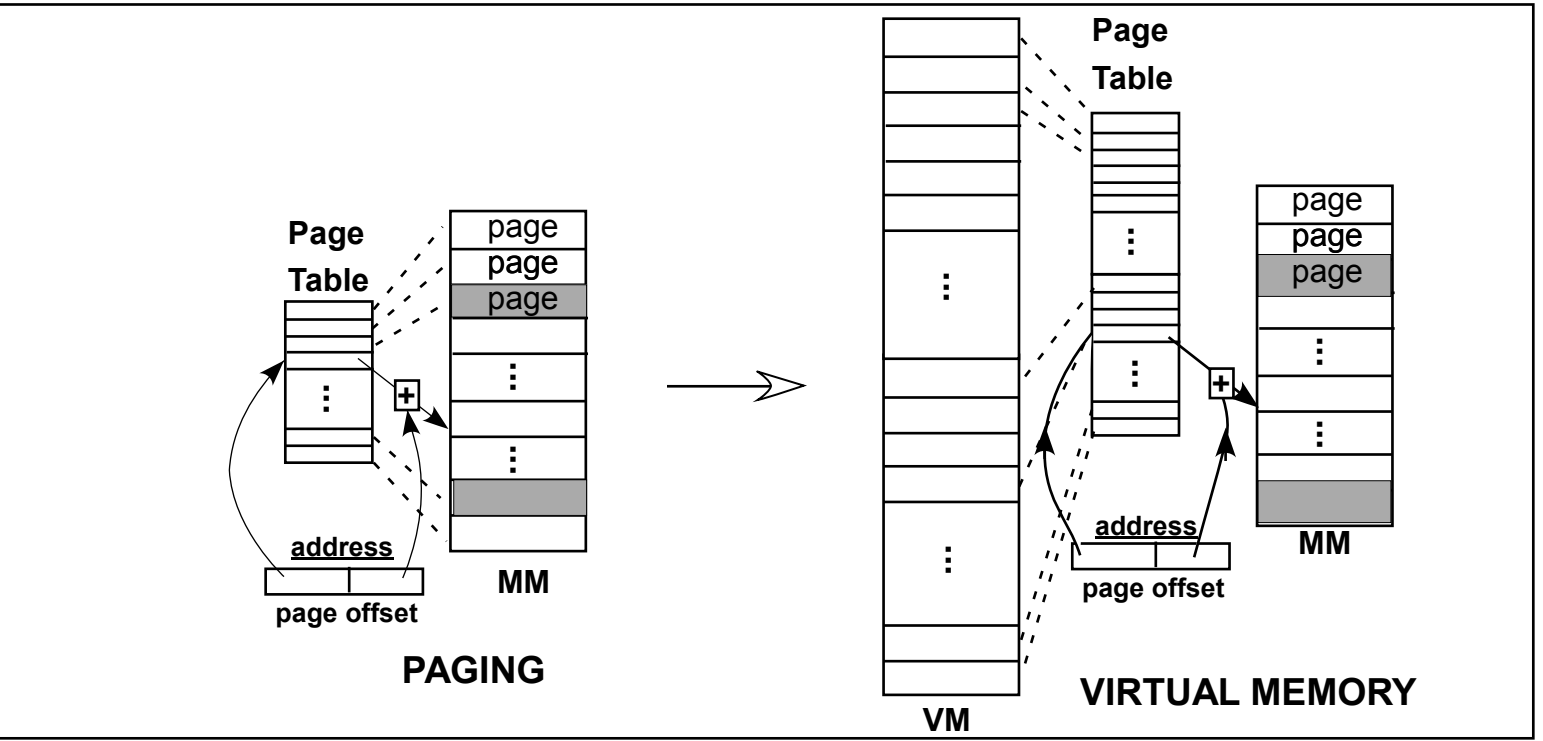
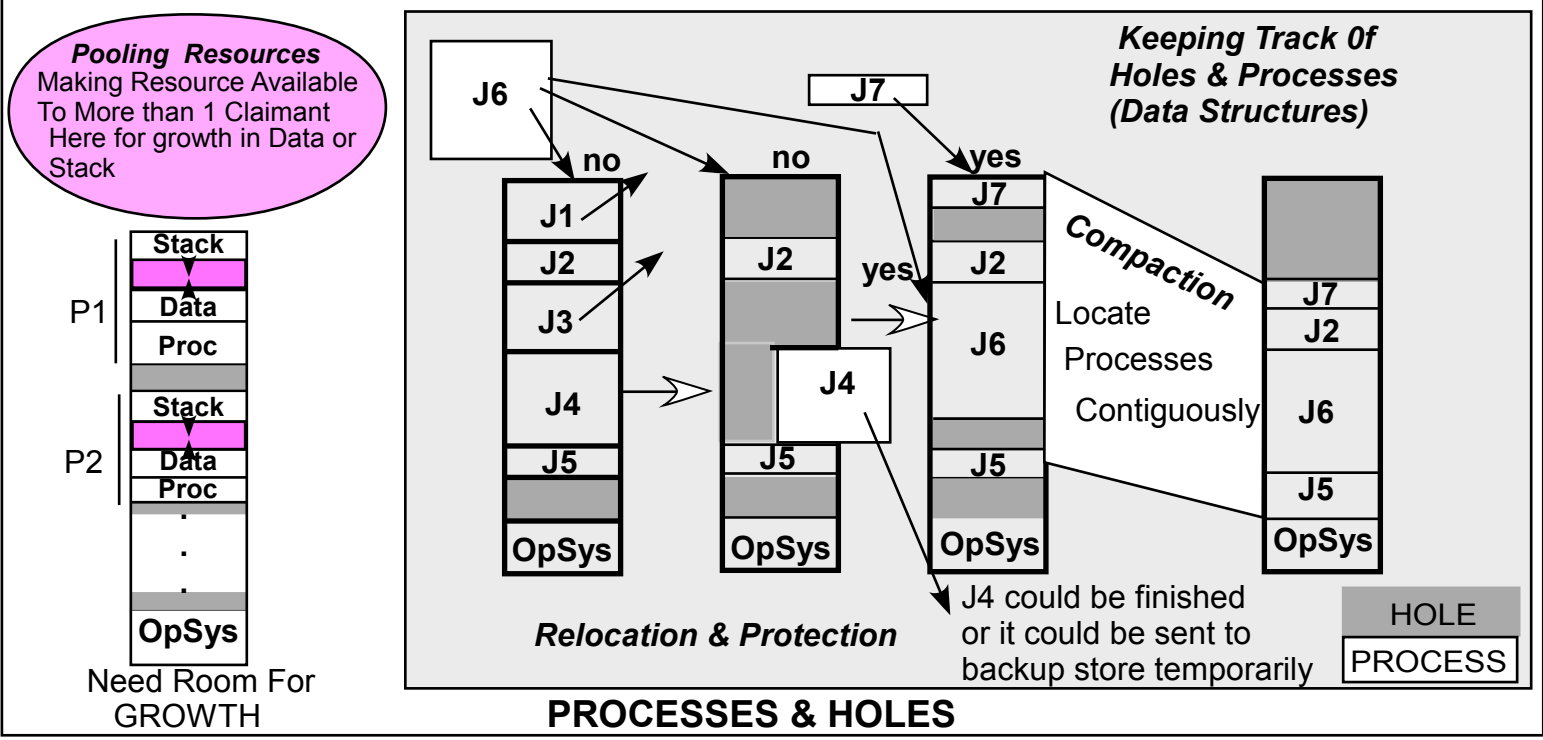
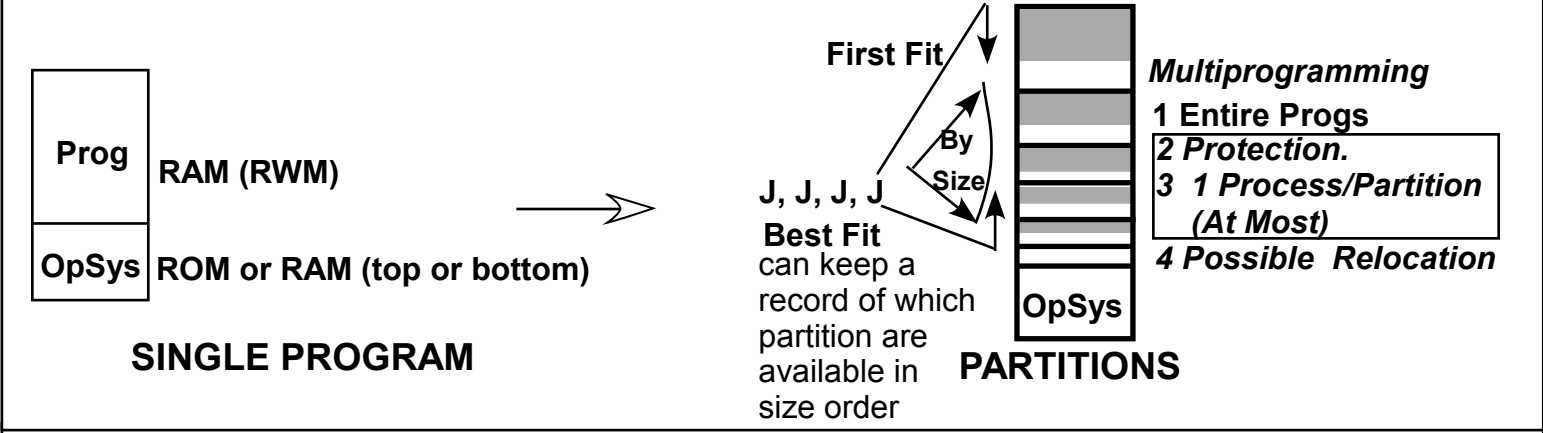
Time a job is in CPU    Time a job is in I/O

Represents the total time a job is in CPU and is in IO, but not the sequence in which CPU and IO occur.  
These times may be sequenced in any order, ex.:

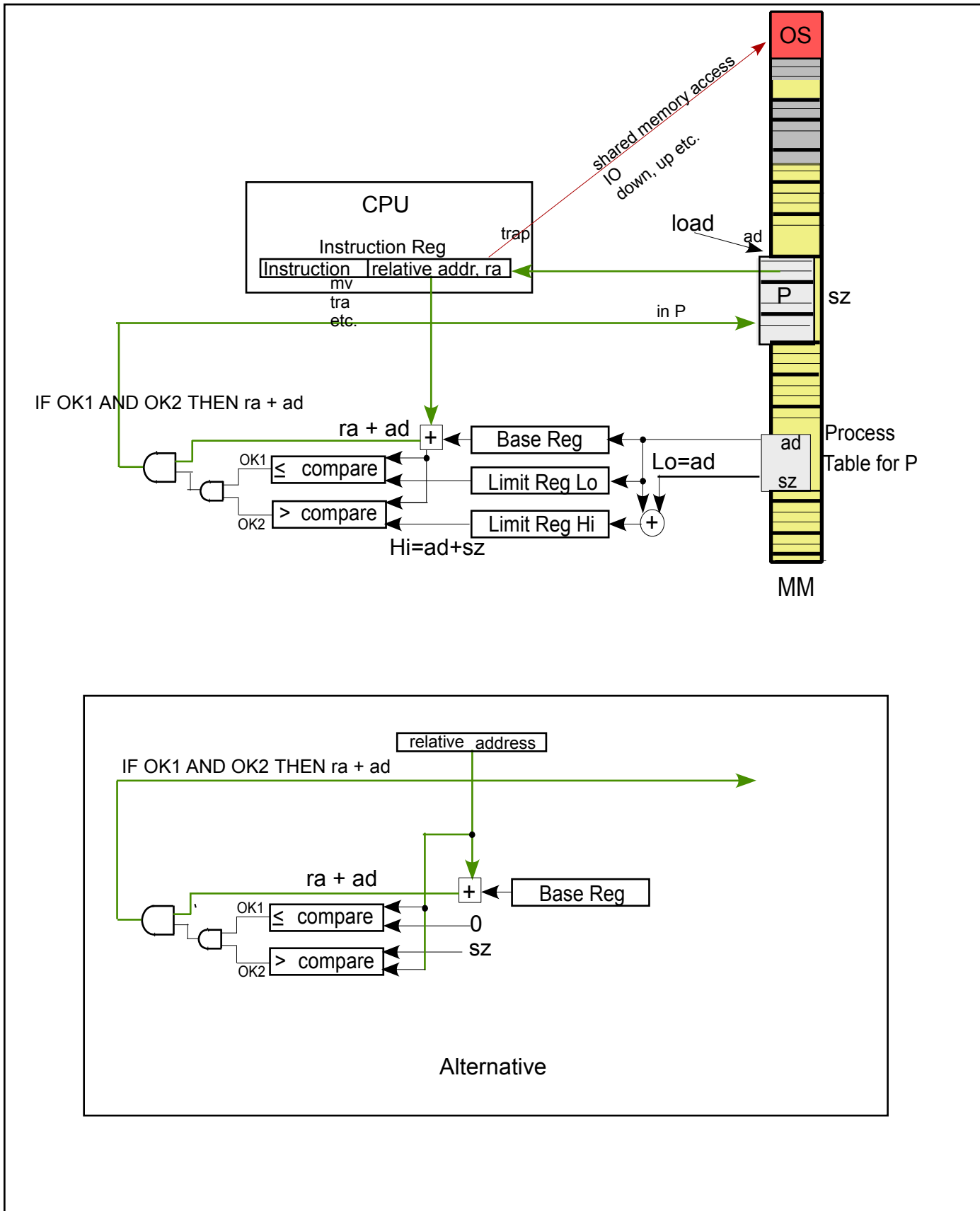


\*\* If 3 jobs are present. How long ( $T$ ) must the jobs be present so that each gets a CPU time of 1.25?  
The fraction of the time devoted to the CPU with 3 jobs present is  $1 - p = .875$ .  
This is shared equally amongst 3 jobs so each gets  $.291$  of the time.  
Therefore  $.291T = 1.25$ . So  $T = 4.3$ . After this there is a job requiring time 1 to completion  
Since there are 2 jobs present we have  $.375T = 1$ , so  $T = 2.67$ . etc.

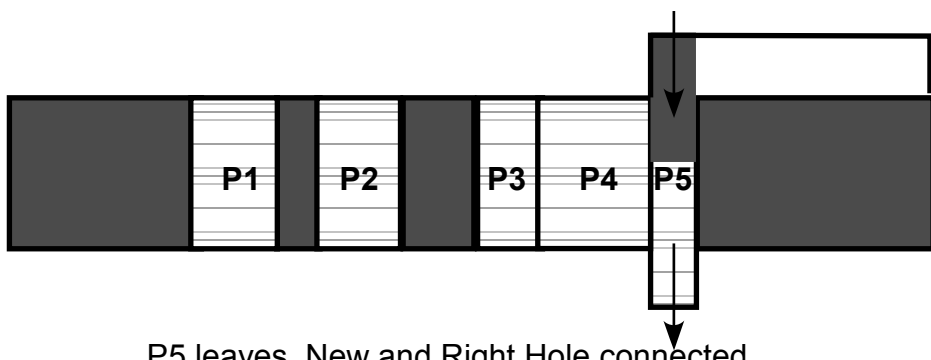
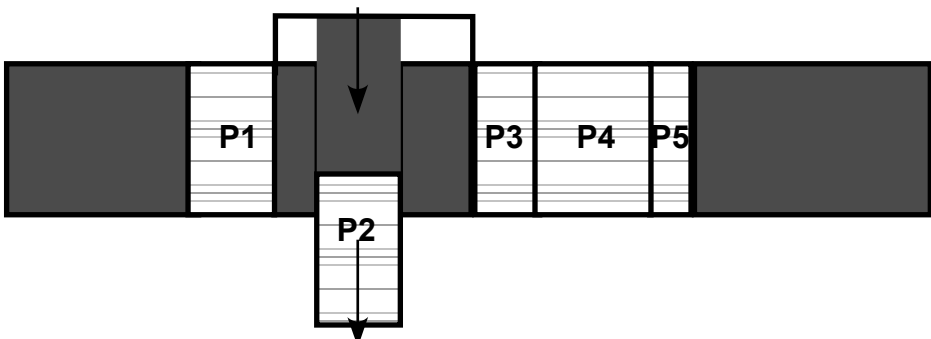
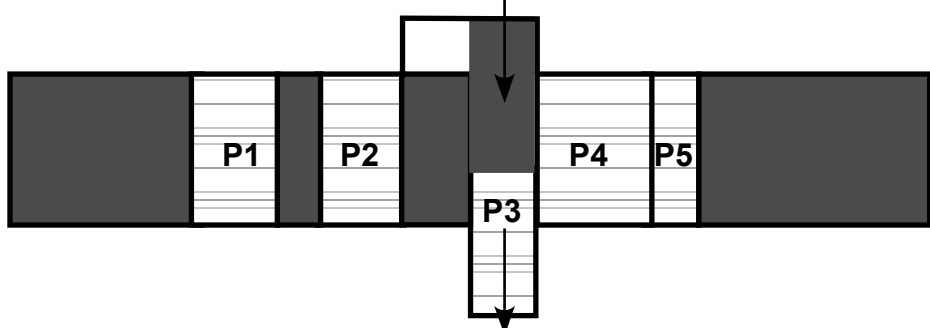
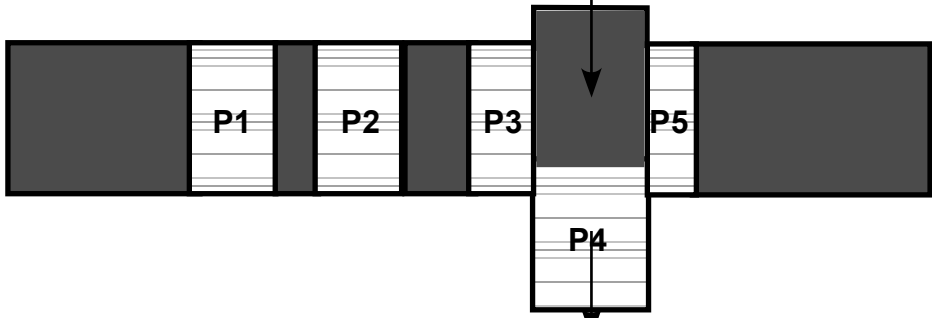
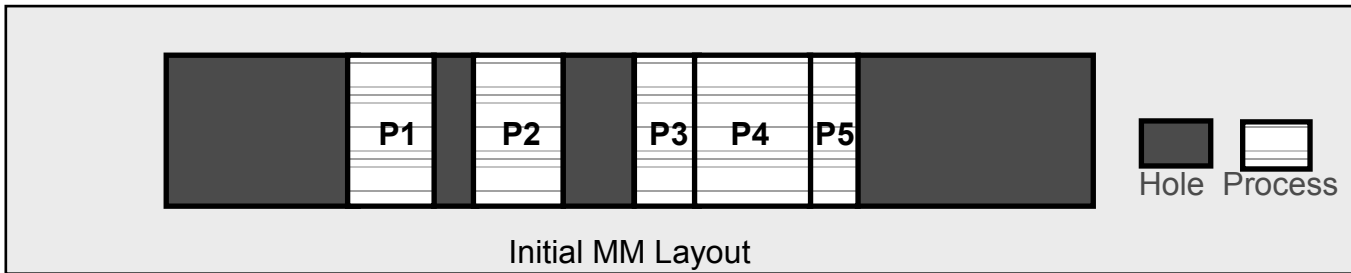
**CPU UTILIZATION USE OF  $1-p^n / n$  TO COMPUTE TIME GIVEN TO SETS OF PROCESSES  
EXAMPLE (BOOK)**



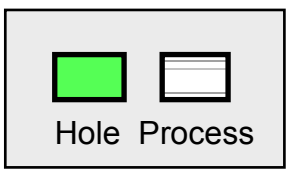
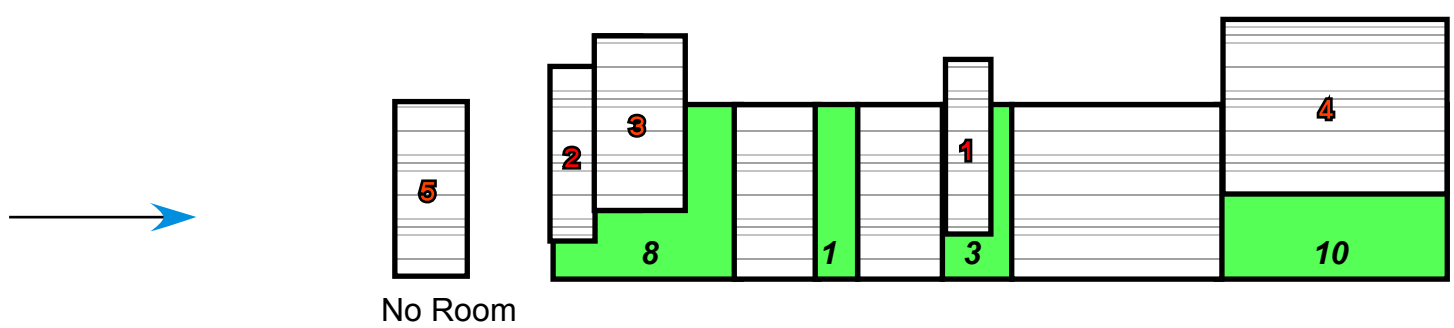
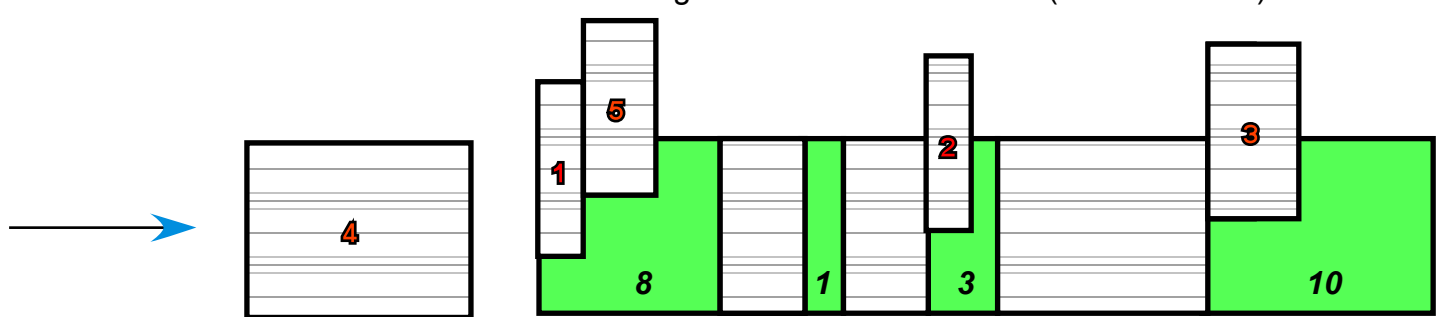
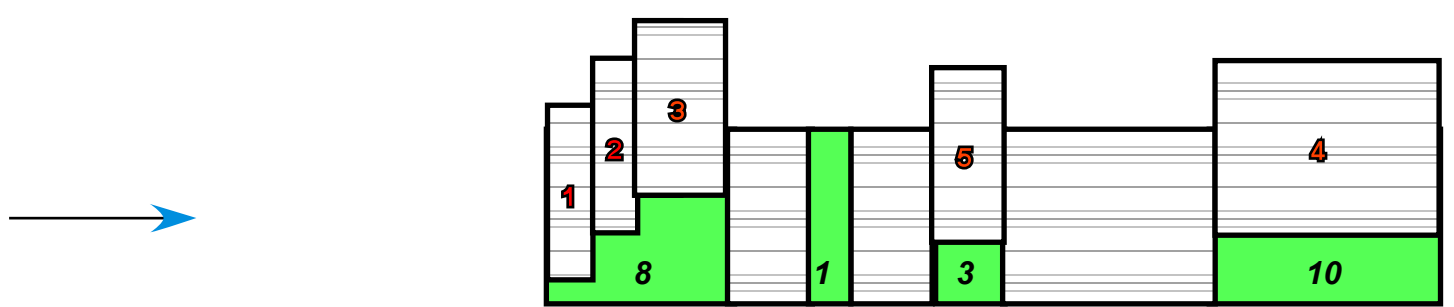
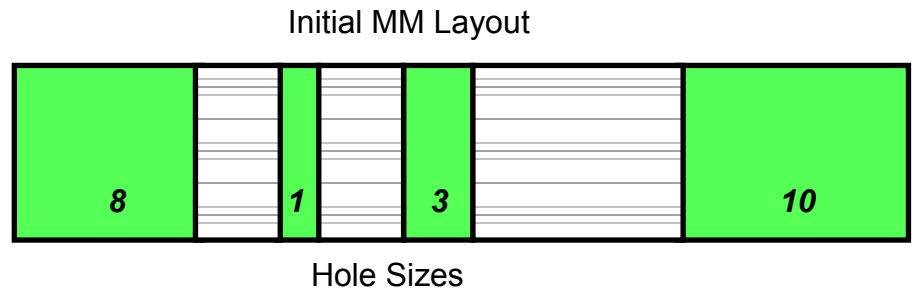
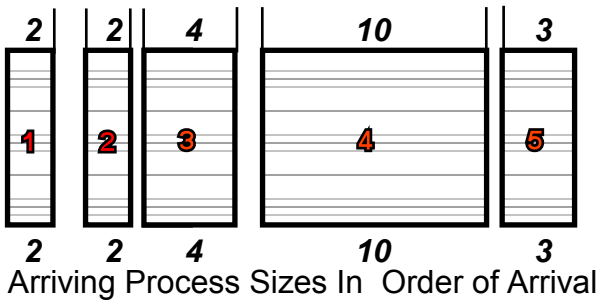
**PROCESS ASSIGNMENT OF MEMORY OVERVIEW**



## RELOCATION AND PROTECTION HARDWARE



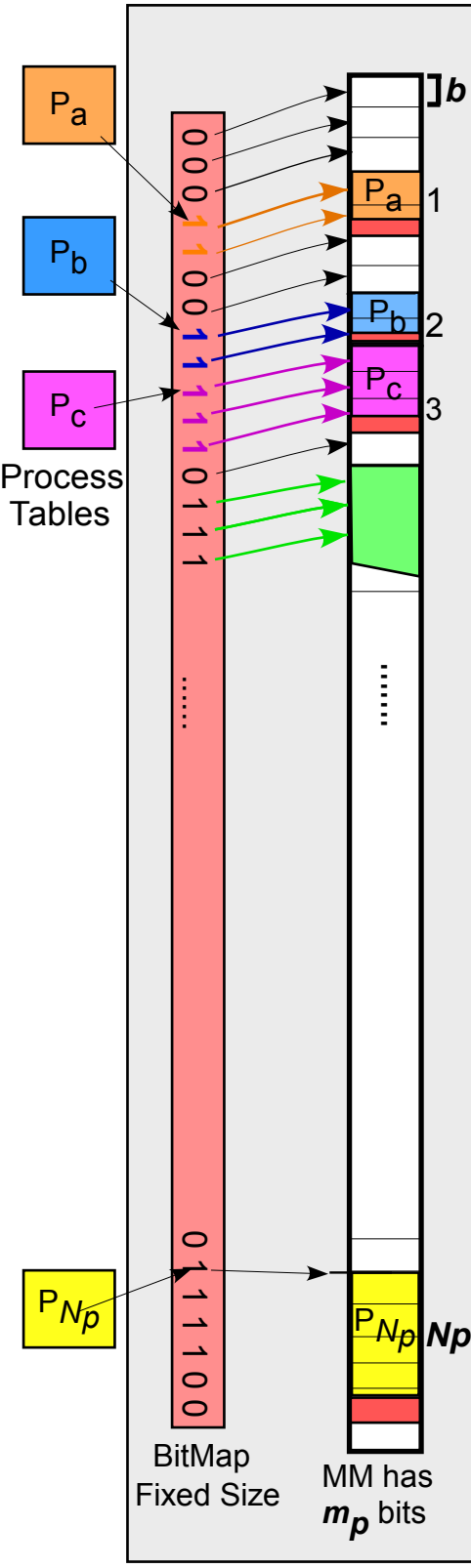
**PROCESSES REMOVAL RESULTANT HOLES**



None Is Uniformly Better Than Another-Best Fit is the Slowest First Fit is the Fastest

**FITTING PROCESSES IN HOLES: FIRST, NEXT AND BEST FIT, SCHEMES An Example**

1. Reconstructing HOLES when PROCESS is Removed very good
2. Finding a HOLE to accommodate a PROCESS fairly bad linear
3. OVERHEAD fixed fairly good



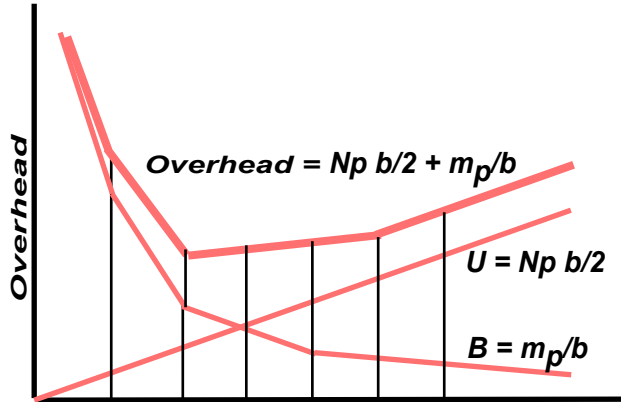
$N_p \rightarrow$  # of procs  
 $b/2 \rightarrow$  bits-lost / proc [on average]

$U = N_p b/2 \rightarrow$  # Procs x bits-lost / Proc =  
 = TOT bits-lost

$b \rightarrow$  bits / block  
 $m_p \rightarrow$  total bits in MM

$B = m_p/b \rightarrow$  # bits in MM Procs / (bits / block)  
 = blocks in MM = bits in Bitmap

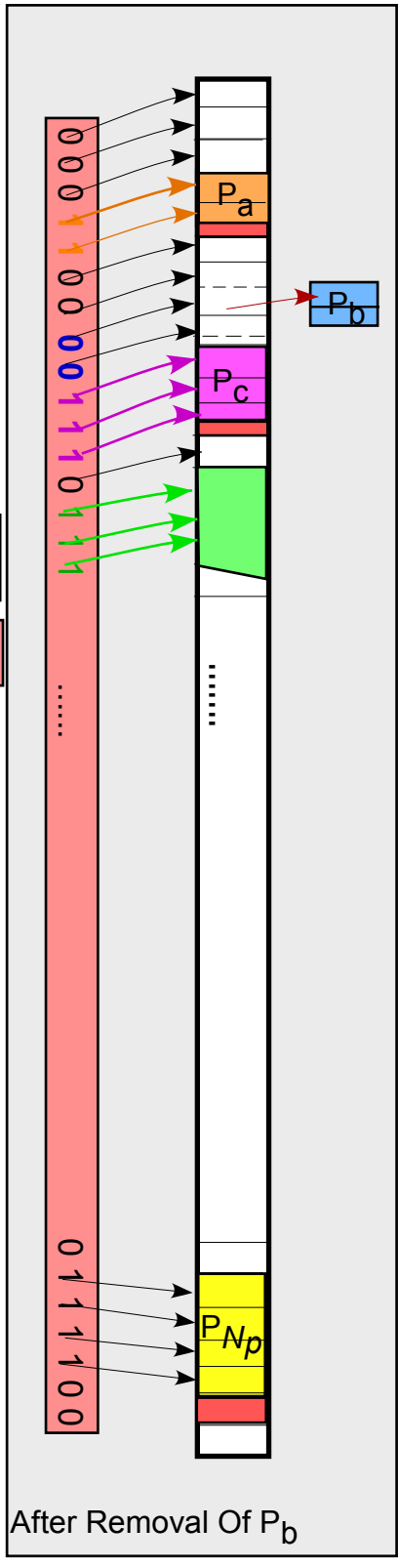
**Overhead =  $U + B = N_p b/2 + m_p/b$**



As = Block Size Increases:  
 $B = m_p/b$  Bitmap Size  
**Decreases**  
 $U = N_p b/2 =$  Loss =  
 Number Of Proc 1/2 Block /Proc  
**Increases**

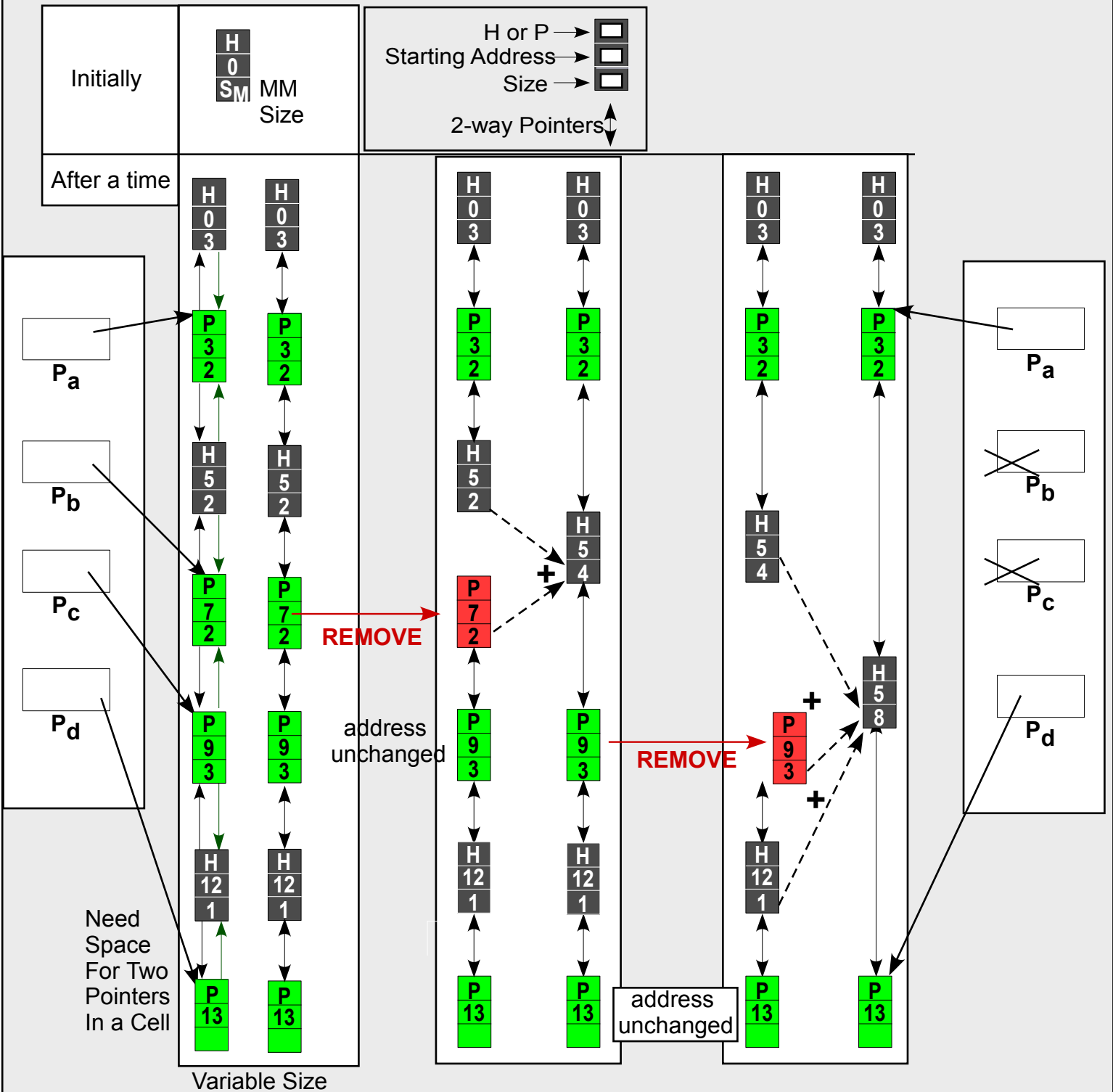
Minimizing:  $0 = d(U+B)/db = N_p/2 - m_p/b^2$

**$b_{optimal} = \sqrt{2m_p / N_p}$**



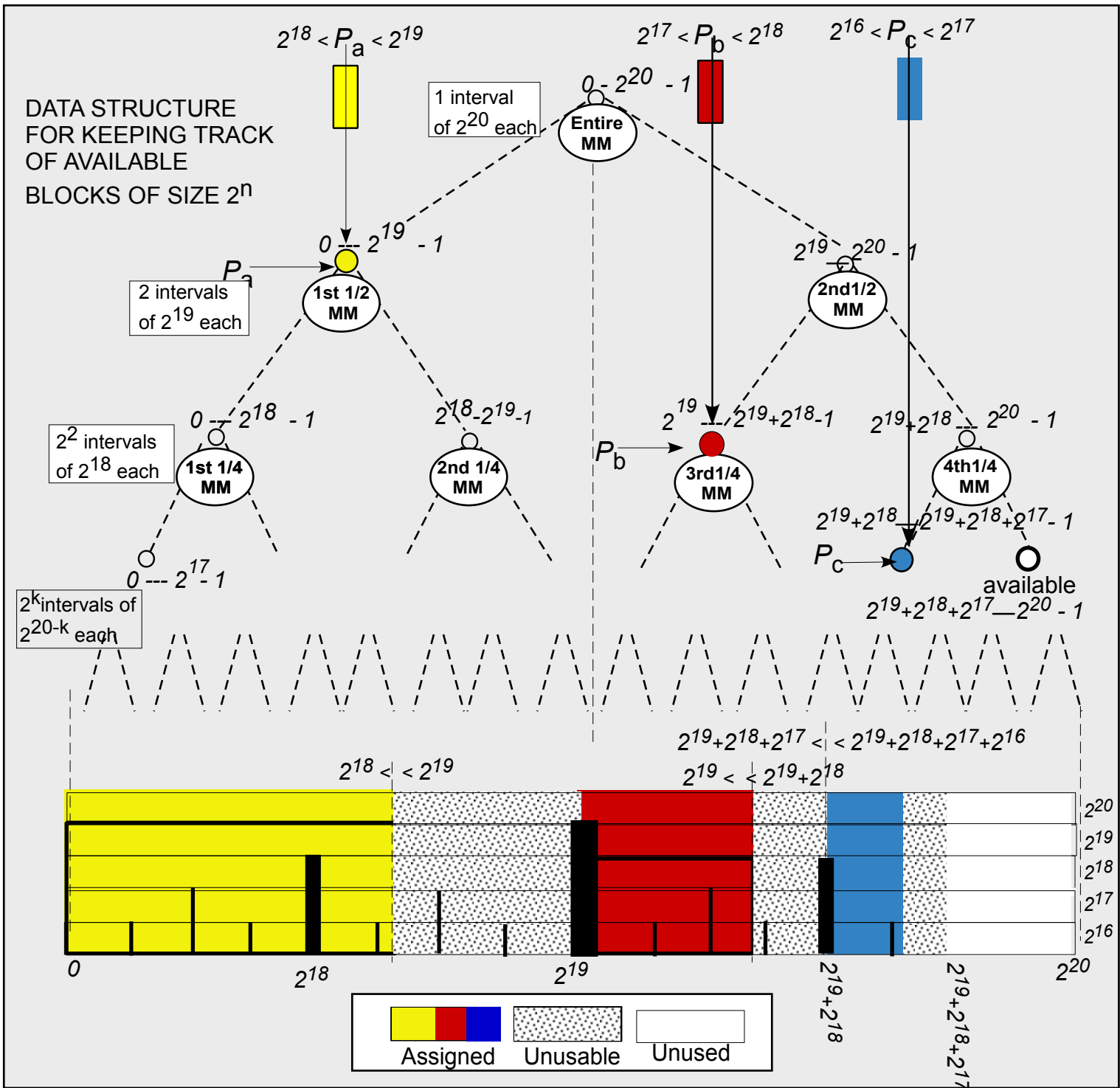
**PROCESSES AND HOLES BIT MAP DATA STRUCTURE \*BLOCK SIZE Analysis\***

- 1. Reconstructing HOLES when PROCESS is Removed good
- 2. Finding a HOLE to Accommodate a PROCESS bad linear
- 3. OVERHEAD ranges from good to worse than Bitmap.



Can Speed Up Search For Hole to Accommodate a Process by having an additional linked list connecting Holes linked list of Holes 2 More Pointers/Cell

### PROCESSES AND HOLES LINK LIST DATA STRUCTURE



The Buddy Algorithm for placing a PROCESS in a HOLE is similar to the following procedure:

To enter a PROCESS, P, of size N, round N up to the next highest power of 2 giving P'. Suppose that the size of P' is  $2^n$ . Now examine the memory. Check for a HOLE of size  $2^n$  starting at location 0, if not, check for such a HOLE starting at location  $2^n$ , if not check location  $2^{n+1}$ , etc. Use Leftmost Depth First Search

In general check for a HOLE of size  $2^n$  starting at location 0 then  $2^{n+k}$  with  $k = 0, 1, 2, \dots$ , until such a hole is first found. P is placed there. The Buddy algorithm is based on a tree data-structure whose use decreases the amount of search for a hole once the level at which a hole is sought has been determined. The algorithm avoids many of the already occupied locations at that level, by **marking** tree nodes of occupied locations and others to help direct the search to an available hole. We first show a simple version of such an algorithm

→ Continued

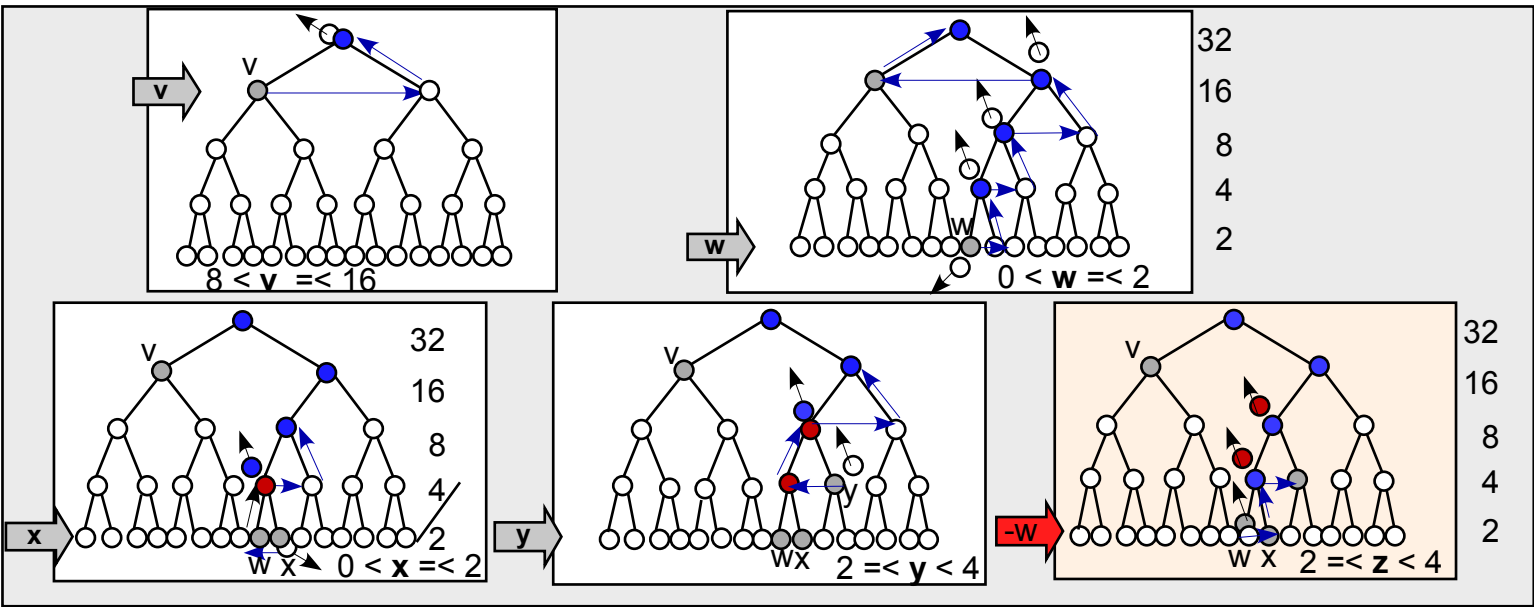
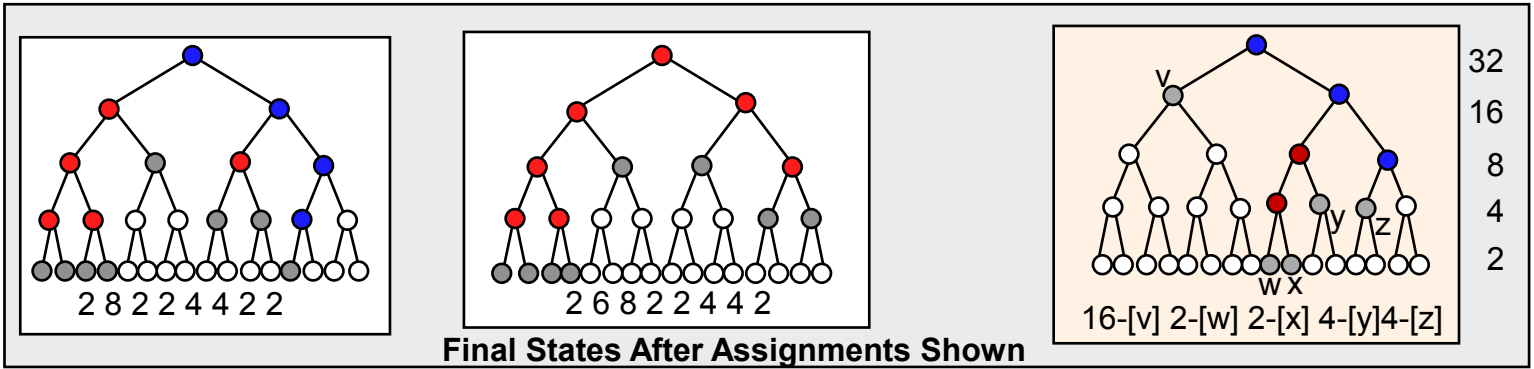
## PROCESSES AND HOLES BINARY TREE DATA STRUCTURE BUDDY ALGORITHM

**Example:**

Here we give a simple algorithm based on the tree data structure for keeping track of Holes and Processes. The algorithm maintains information (MARKS) at the nodes of the tree to shorten the search for a hole which will accommodate a given Process. in an MM contains  $2^5$  bytes in this example. So if the size of process  $P_s=12$ . The next highest power of 2 is  $16=2^4$ . Go, by **leftmost depth first search** to level  $2^4$ . Does the leftmost node at that level indicate there is space starting at location 0? If so place P there and mark the tree node accordingly. If it is not there look at the node corresponding to location  $16 = 2^4$ , In general start at 0 then  $2^{4+k}$   $k = 0, 1, 2, \dots$ , until such a hole is first found. P is placed there. If no node at level  $2^4$  is found P cannot be placed in MM. The information that is maintained at each node is given by the following node "Markings"

○ unoccupied available  
 ● unoccupied not available, some below available below  
 ● unoccupied and not available-none below available  
 ● occupied

**MARKS**



Notice; not only can considerable space be unusable because Processes do not generally have sizes which are a power of two, but the placement choice for a Processes which can fit in a number of holes can effect the loss of space in the MM. Processes of size  $2^k$  equal in number to the number of size  $2^{k+1}$  partitions in the MM could be located 1 per  $2^{k+1}$  partition. Then there would be no room for a Process of size  $2^{k+1}$ . Whereas if these Processes were located with 2 in each of the leading  $2^{k+1}$  partitions there would be plenty of room for Processes of size  $2^{k+1}$ . So generally it is advantageous to fill a partially filled partition when 1/2 of it is available rather start an empty partition. So one ought to place a Process at the lowest address at which it can be accommodated. Leftmost depth-first search would achieve this- Backup for this search could be minimized if the algorithm chosen gives the highest level available by both the left child, and the right child. Such an algorithm is developed now.

**NODE NUMBERING**

$P'_L = \max(*C_{LL}, C_{LR})$

$P'_R = \max(C_{RL}, C_{RR})$

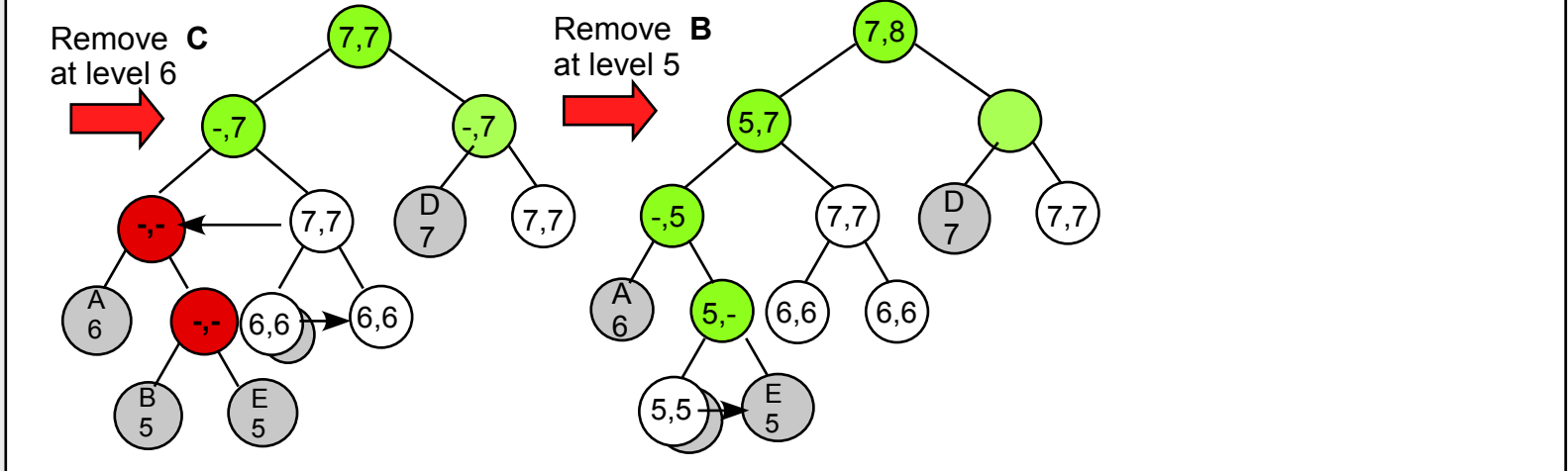
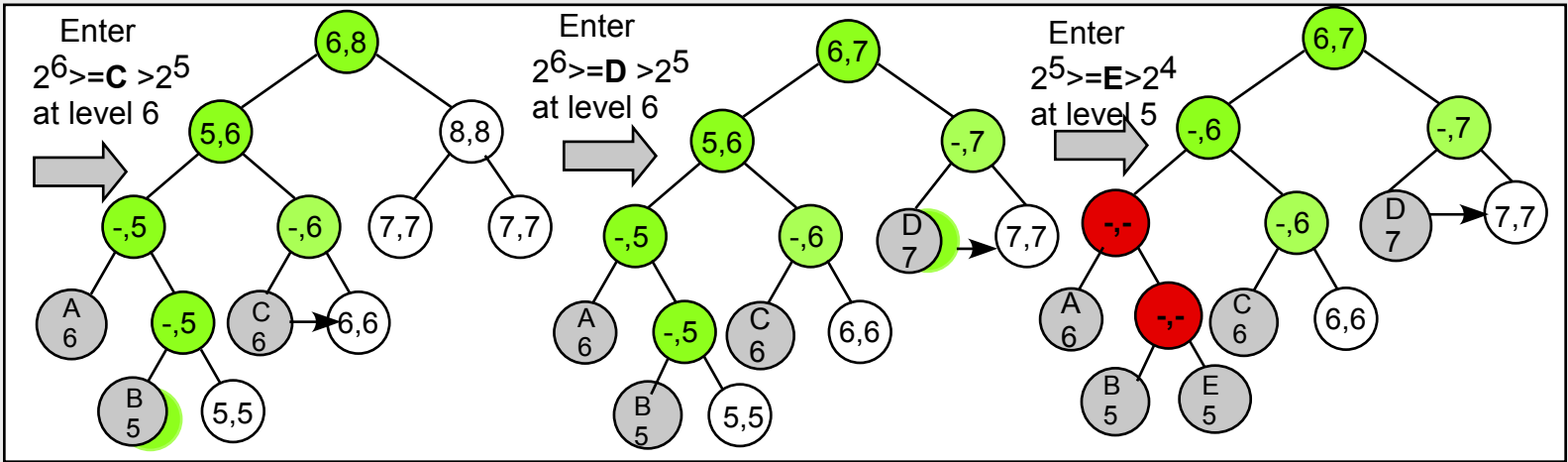
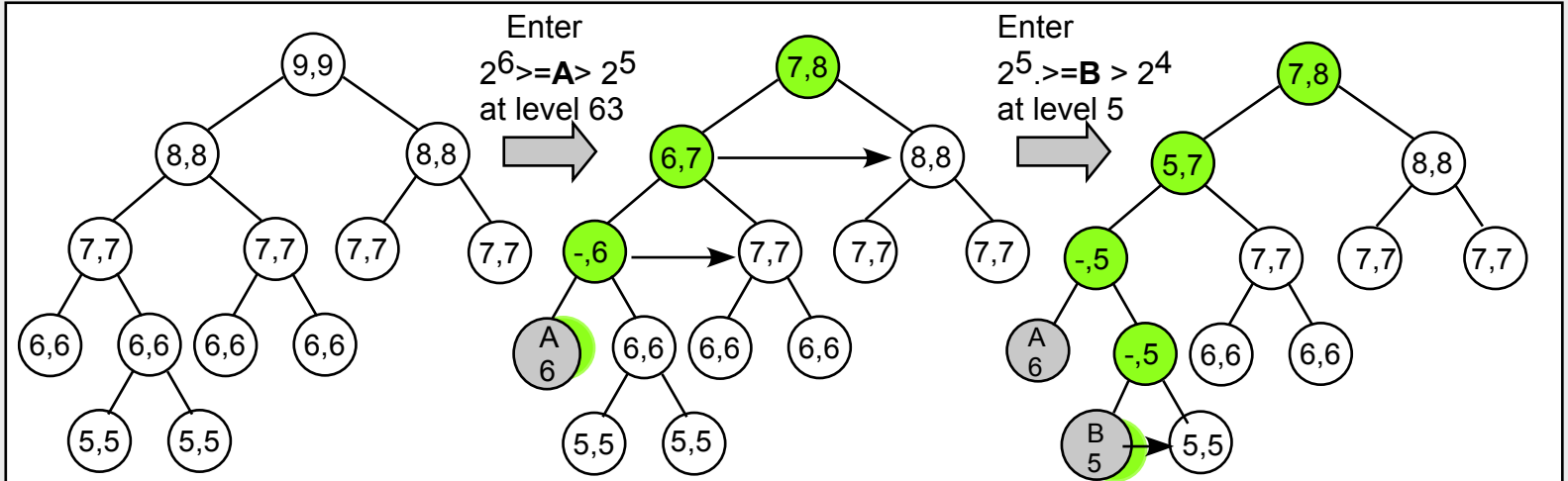
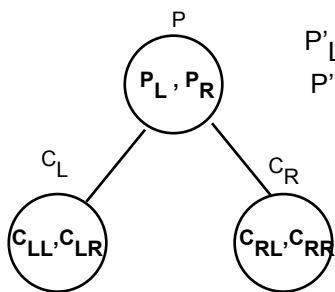
if  $(P'_L = P'_R = \text{the level of Ps child})$   
 then  $\{P_L = P'_L = \text{the level of P}\}$   
 else  $\{P_L = P'_L \text{ and } P_R = P'_R\}$

\* $C_{xy}$  is either =

- 1) the deepest level reachable of C itself if C is not marked or
- 2) the deepest level reachable from C, with the first move to its x (L or R) child, if a node below C is occupied the highest level reachable from C at which the node is unoccupied.

**MARKS**

- grey = occupied
- green = not occupied descendant occupied
- un-occupied & reachable
- red = not occupied no unoccupied descendant reachable



Because it is generally best to use siblings of already occupied node it is probably best to pursue the paths whose highest available level is lowest, but still  $\geq$  level sought..

**BUDDY ALGORITHM MARKING TREE NODES TRACES**

## After A Time the Number of Holes in Memory is 1/2 the Number of Processes

There are  $n$  processes. There are  $n+1$  positions between successive processes each of which could be occupied by a hole or *nil* (nothing). The average layout of MM can then be represented as :

(*) Hole or $\lambda$ #	1	2	3	...	$n$	$n+1$
	* 1	* 1	*	...	* 1	*
(1) Process #	1	2	...		$n$	

Each \* is either a hole (0) or unoccupied ( $\lambda$ ).

Each \* is either a hole (0) or unoccupied ( $\lambda$ ). Considering all possible combinations of assignments to the \*s. In the average assignment 1/2 of the \*s are 0. So there are  $n$  1s (processes) in each and an average of  $(n+1)/2$  0s (holes) per combination. Therefore:

Average number of processes =  $P(n) = n$

Average number of holes =  $H(n) = (n+1)/2$

So the ratio of Holes to Processes is  $H(n) / P(n) = (n+1) / 2n = (1 + 1/n) / 2$

$= 1/2$  quickly as  $n$  increases

1 represents a process  
 0 represents a hole  
 $\lambda$  represents an empty  
 position

*	1	*	1	*
0	1	0	1	0
0	1	0	1	$\lambda$
0	1	$\lambda$	1	0
0	1	$\lambda$	1	$\lambda$
$\lambda$	1	0	1	0
$\lambda$	1	0	1	$\lambda$
$\lambda$	1	$\lambda$	1	0
$\lambda$	1	$\lambda$	1	$\lambda$

average number of 0s =  $12/8 = 3/2 = (n+1) / 2$

**Example  $n = 2$**

### Dependence Of Utilization On The Ratio of Process to Hole Size

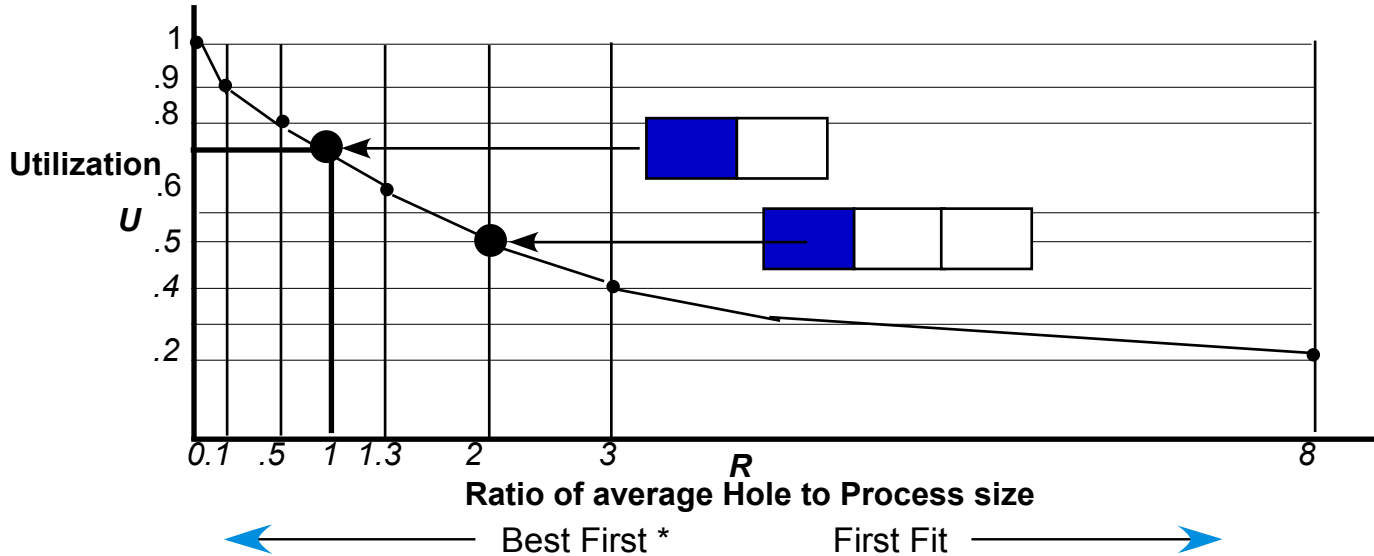
Obviously, if the ratio of hole size to process size decreases, and the number of holes is the same as the number of processes, the fraction of the memory occupied by holes decreases, that is Utilization improves (less of the memory is devoted to Holes). In fact if the number of holes bears any fixed relation to the number of processes decreasing the ratio of hole to process size will increase the Utilization. In fact once the size relation is fixed Utilization dependence on ratio of sizes is completely determined. Since a fixed relation between the number of Holes and Processes (Number of Holes = 1/2 Processes) is known, how the Utilization will increase with the decrease in the ratio of Hole to Process size is determined.

Process size is unaffected by whether we use the Best First or First Fit algorithms, but Hole size tends to be smaller with Best Fit than First Fit. (This assumes that the 1/2 rule holds equally well independent of which Best or First Fit is used) The details are on the next page.)

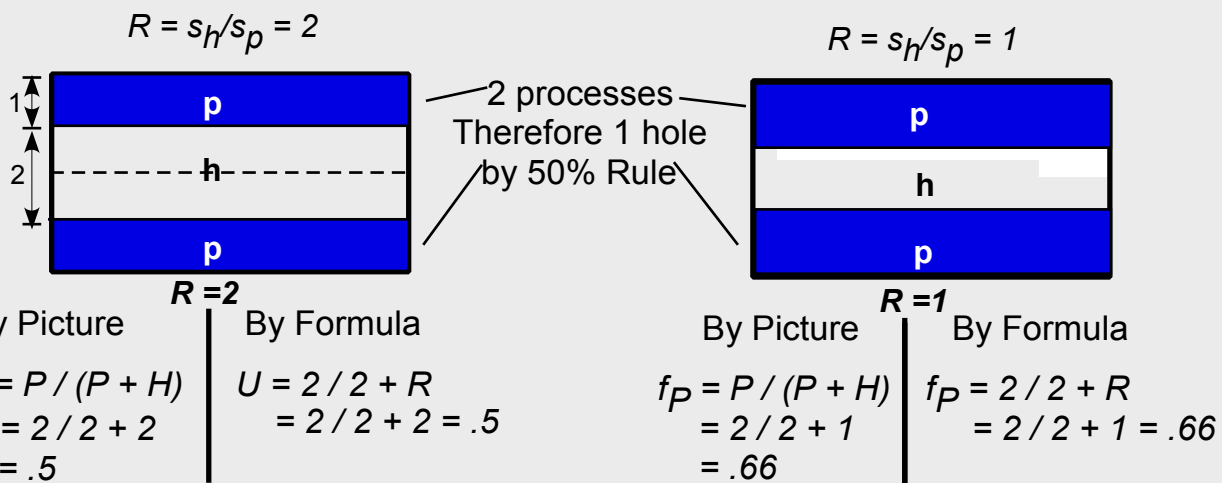
## PROCESSES AND HOLES-THE 1/2 RULE

$s_h$  is the average size of a Hole in pages  
 $s_p$  is the average size of a Process in pages  
 $n$  is the number of Processes in MM  
 $U$  fraction of the MM in Processes = **Utilization**

$$\begin{aligned}
 U &= s_p n / ( s_p n + s_h (n/2) ) \\
 &= 1 / ( 1 + s_h / 2 s_p ) \\
 &= 1 / ( 1 + R/2 ) \quad \text{- where } R = s_h / s_p \quad \text{Ratio of average Hole to Process} \\
 &\quad \text{size} = 2 / 2 + R
 \end{aligned}$$



\* Best First always places a new entering Process so as to leave the minimum size hole  
 (Note an exact fit is always Optimal.) This gives good Utilization, but tends to leave (many) small Holes which become unuseable. Neither First or Best Fit is uniformly optimal.



$P$  and  $H$  are the total memory devoted to Processes and Holes respectively

Using the 1/2 rule some idea about utilization's dependence on the ratio average hole to average process size

## PROCESSES AND HOLES UTILIZATION ANALYSIS USING THE 1/2 RULE