

Paging works because of Locality of Reference. Good Replacement Algorithms minimize frequency of page faults

2 SIGNIFICANCE OF REPLACEMENT ALGORITHMS

Basic Replacement Algorithms and Variants which balance efficient replacement and algorithm complexity

3 REPLACEMENT ALGORITHMS EXAMPLES: FIFO (QUEUE, CIRCULAR), OPT

4. PAGE INPUT SEQUENCE, REPLACEMENT ALGORITHMS-SECOND CHANCE FIFO,

5 BASIC PLANS: FIFO (QUEUE, CIRCULAR), LRU, OPT,

Another Variant

6 REPLACEMENT ALGORITHMS- AGING, WORKING SET (APPROX)

Hybrid Approaches (Practical)

7 1-2 Handed Clock Algorithm Based On Second Chance-FIFO

Clever, but expensive true LRU Replacement algorithm

8 REPLACEMENT ALGORITHMS TRUE LRU- MATRIX IMPLEMENTATION

9 EFFECTIVENESS OF CLOCK, AGING ALGORITHMS THE WORKING SET ALGORITHM

A Variety of Working Set Replacement Algorithms

10 TIME INTERVAL APPROXIMATION WORKING SET ALGORITHM IN CLOCK ALGORITHM CONTEXT 1.

11 TIME INTERVAL APPROXIMATION WORKING SET ALGORITHM IN CLOCK ALGORITHM CONTEXT 2.

12 DIRECT APPROXIMATION WORKING SET ALGORITHM 1

13 BIT MAP APPROXIMATION WORKING SET ALGORITHM 2

14 REPLACEMENT ALGORITHMS ACTUALLY USED

Algorithms guaranteed to improve performance as the number of pages increases

15 STACK REPLACEMENT ALGORITHM

16 STACK REPLACEMENT ALGORITHM- LRU IS ONE

17 STACK REPLACEMENT ALGORITHM-OPT IS ONE TOO

Notes on Unix's method for managing page free lists and dirty pages

18 UNIX DEMAND PAGING: DAEMONS, SWAP LIST, PAGE MAP

19 DAEMONS, CRON, PRINTER, MAIL

Interesting relation between data structures

20 THE EQUIVALENCE OF FIFO QUEUE AND CLOCK DATA STRUCTURES

CONTENTS

PAGE REPLACEMENT ALGORITHMS:

Good Page Replacement Algorithm Means Efficient Computing

In multiprogramming, **efficient utilization** requires that **many Process pages** always **be present in MM**. This implies traffic in **pages moving in and out of MM**. Out because, when a page not in MM, P_{new} , is addressed it is a **Page Fault**- i.e P_{new} must be brought in from Disk or Disk Buffer, but here may be no room in MM without replacing some page, P_{old} , currently in MM. If process P_{old} has been modified since its arrival in MM, it must be written back to the **backup disk**. This together with finding P_{new} on the disk (or disk buffer) and overwriting the page frame made available by P_{old} 's departure is a **time consuming operations**. Therefore in choosing a page to replace, the system attempts **to find the one that is least likely to be needed soon**, or the one that will be needed again after the greatest time. It can only do this based on information it has about **previous page requests** (though perhaps some programmer or compiler prepared predictive information could accompany each page about the pages it is likely to request next). The efficiency of the algorithm that chooses the page to replace is also of concern.

Replacement Algorithms: Memory Access:Choice Slow Larger or Faster Smaller Memory.

Though we speak mostly of the MM - Backup Disk case, Replacement Algorithms have similar significance in all memory systems in which there is a Slow Large Memory and a Faster Smaller Memory. Similar replacement algorithms are considered for all these cases although the speed of execution may vary in importance.

Locality Of Reference

Paging works because of the empirically verified phenomenon of "**Locality Of Reference**". That is, the address requests issued by an active page tend to remain in that page and when leaving that page to remain within a small set of pages, the "**Working Set**", for a considerable number of instruction executions. As the number of requests increase **the working set changes**, adding and dropping pages, but generally only **very slowly**. So it is important when replacing a page of a Process which has not yet completed to only **replace those pages unlikely to be in a Working Set** so that they will not have to be quickly brought back into MM.

The slowly changing Working Set is a strong determinant in the design of replacement algorithms. That is, the **recent past use of a page is assumed to be predictive of its future use**. Small variations in the recency of use is insignificant. As always there is the sometimes opposite influences of the cost, time, space, hardware, of implementing such algorithms.

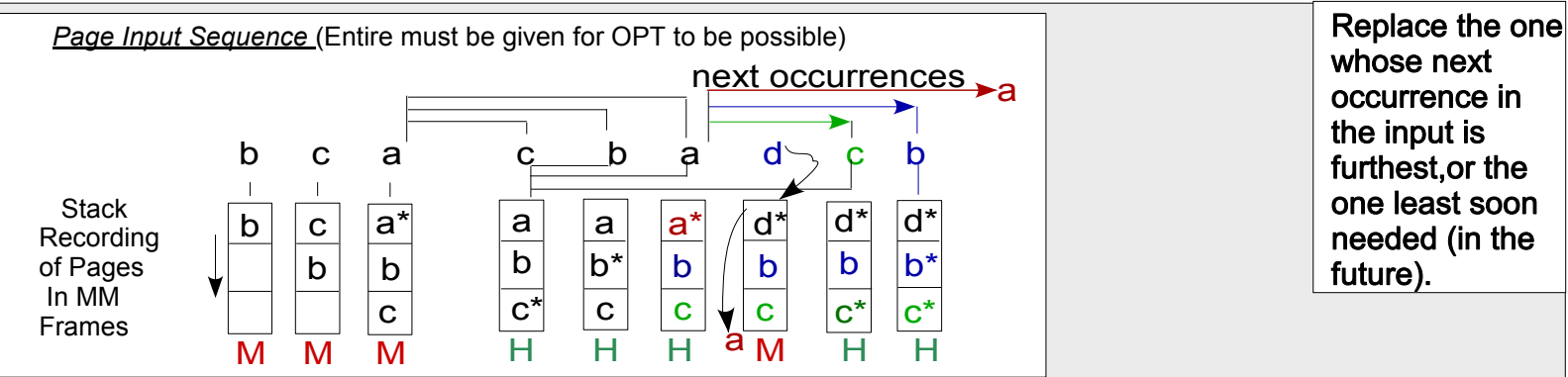
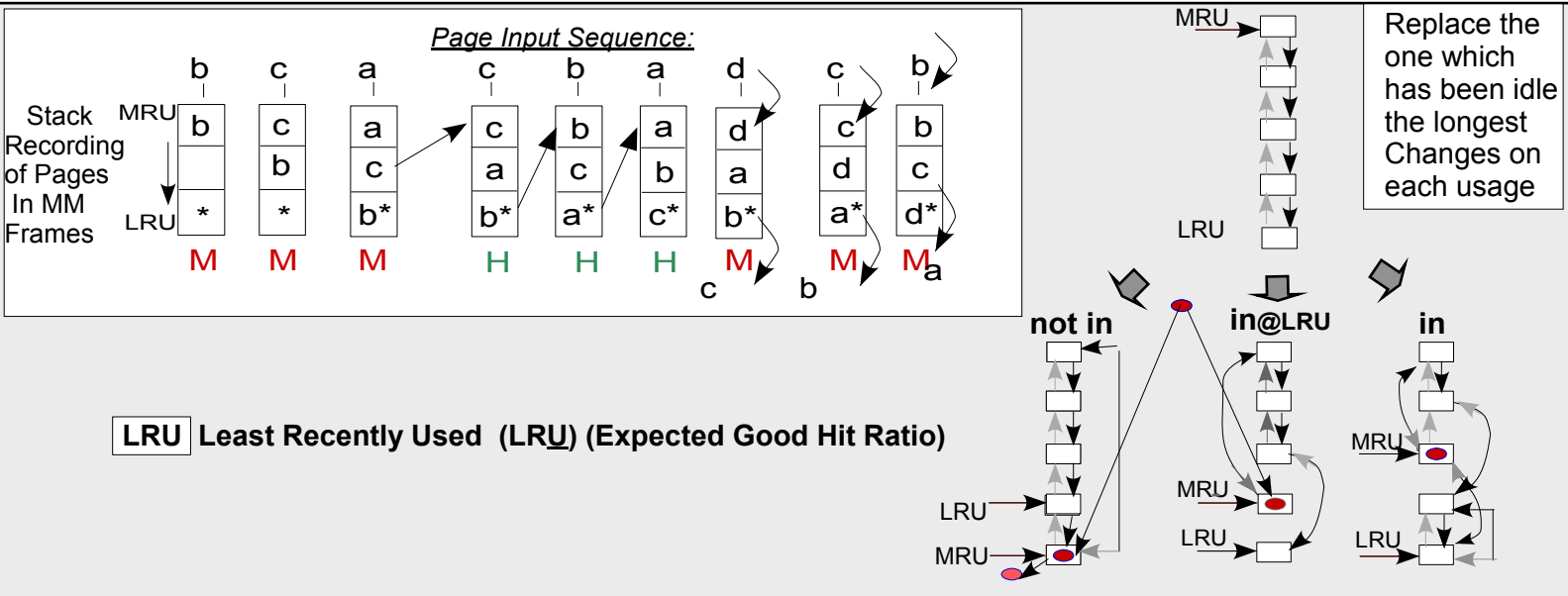
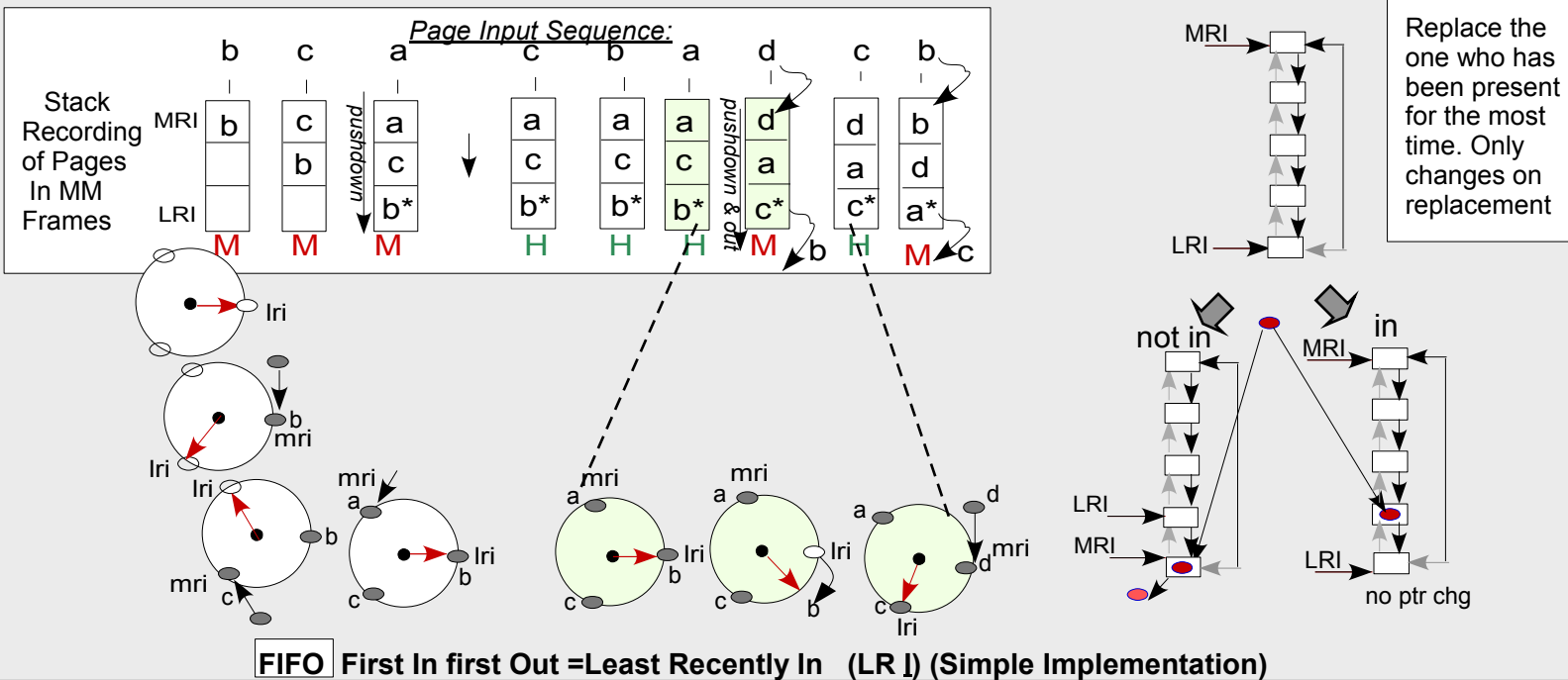
Page Replacement Algorithms are Tested By Simulations On Sample Page Request Sequences.

A proposed replacement algorithm is tested by a simulation based on a sequences of page requests actually encountered in running Processes or based on probabilities compiled from such runs. It is useful to **know the best that could have been done for a given sequence of requests** in order to compare it with the performance of a proposed algorithm. Having the entire sequence of requests available before the test allows such a best possible result to be found.

MM Size and Replacement Algorithms

If information units (page, block) are equally distributed over MM1 of size s , and MM2 of size S then for given access times of MM1 and of MM2, the probability of finding an information unit in MM1 is proportional to s/S . Replacement Algorithms are designed to increase that probability. MM Cache Replacement algorithms importance is proportional to the ratio of the size of MM and the MM cache.

SIGNIFICANCE OF REPLACEMENT ALGORITHMS



Same Page Input Sequence For All Three Pages.
Pages Leaving are Not Simulated

H A Hit in the page input sequence
M A Miss in the page input sequence
* **Dispensable Block** = The block to be replaced if replacement is needed

REPLACEMENT ALGORITHMS EXAMPLES: FIFO (QUEUE, CIRCULAR), LRU, OPT

A Page Input Sequence (PIS) only records a page input when the page addressed changes. The hit ratio under this input sequence is H_p . If it is assumed that the average number of successive accesses in a single page, n_a then the actual reference hit ratio H can be computed.. On each hit of the PIS there will in fact be, n_a successive accesses to that page. On a miss of PIS there will be n_a-1 (the first being a miss.) successive accesses. So given PIS, H is computed as follows:

N_p = the total number of pages in the page input sequence.
 H_p = total # of pages in page input sequence that are Hits
 n_a = the average number of successive accesses to the same page in the page reference sequences
 H = Hit Ratio = total # of pages of page reference sequence that are Hits / total # of page references of page reference sequence

$$H = [H_p N_p] n_a + [(1 - H_p) N_p] (n_a - 1) / N_p n_a$$

$$H = H_p n_a - H_p n_a + n_a - 1 + H_p / n_a$$

$$H = (n_a - 1 + H_p) / n_a$$

$$H = 1 - (1 - H_p) / n_a = 1 - (M_p / n_a)$$

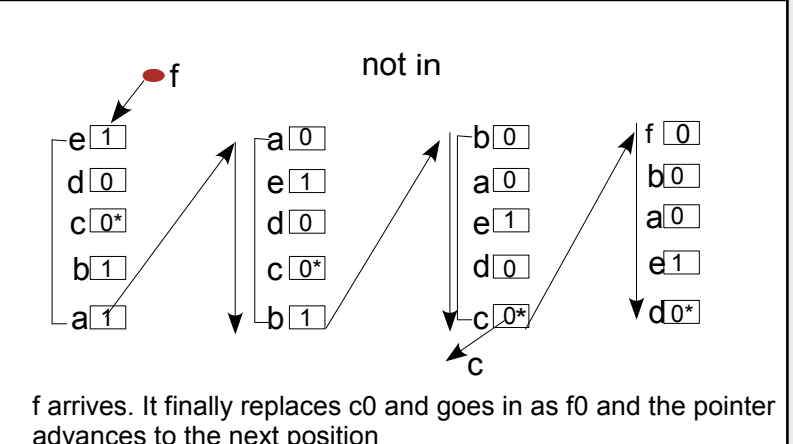
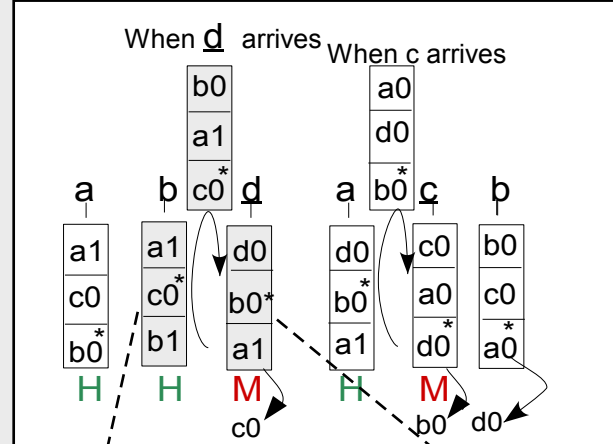
if $n_a = 4$ then the number of Hits =

3	3	3	4	4	4	3	4	4
b	c	a	c	b	a	d	c	b
M	M	M	H	H	H	M	H	H

Page Miss Ratio = $M_p = 4/9$, $n_a = 4$
 $H = 1 - (M_p / n_a) = 1 - (4/9)/4 = 1 - (1/9) = 8/9$

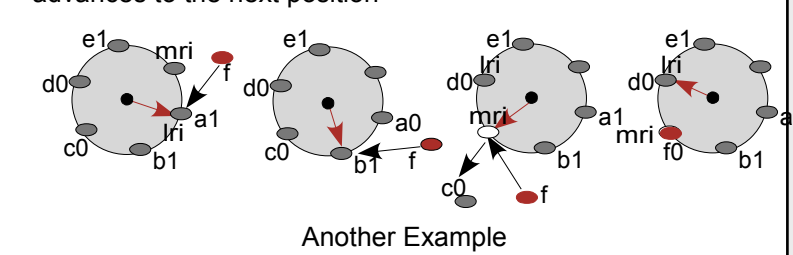
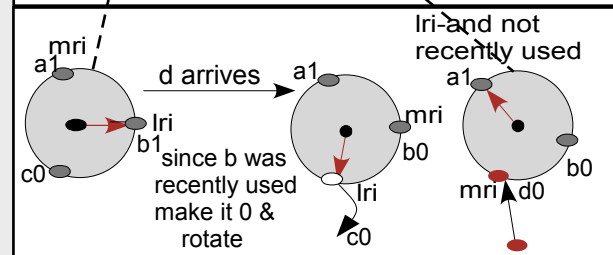
$H = \text{Total Hits} / \text{Total Trys} = [N_{hits} \times n_a + N_{miss} \times (n_a - 1)] / [N_{hits} + N_{miss}] \times n_a$
 $H = \text{Total Hits} / \text{Total Trys} = [5 \times 4 + 4 \times 3] / [5 + 4] \times 4$
 $H = \text{Total Hits} / \text{Total Trys} = 32 / 36$
 $H = \text{Total Hits} / \text{Total Trys} = 8 / 9$

If a page is referenced mark it 1--Choices:
 Either: 1) **Periodically:** Make all 0
 Or: 2) **On Replacement Scan:** 1. Make page reference mark 0 when 1 is detected
 In this case it is called the **1 Handed Clock Algorithm.**



Legend for reference marks:

- in: ●
- no change:



SECOND CHANCE FIFO (NOT RECENTLY USED)
 (Instead of just using 1 bit (reference) to determine 2nd chance Use 2 bits: <referenced, modified>
 <0,0> <0,1>, <1,0>, <1,1> to determine 2nd chance)

Basic Plan For FIFO Main Program

```

Frames[0:N-1]
first_in= 0
Input Pages[1:M-1]
Read (Input Page (k=0 to M-1))
Search( Frames(i=0 to N-1), Input Page[k] )
{at i:
  if empty: then incr(Miss), Frames[i]<--- Input Page[k]
  if found: then incr(Hit)
}
Not found: (all non-empty)then
  incr(Miss) and
  Replace(Frame[first_in], Input Page[k])
  first_in = (first_in+1)modN
  
```

Basic Plan For LRU Main Program

```

Frames[0:N-1]
Input Pages[0:M-1]
mru=0, lru=0;
Read (Input Page (k=0 to M-1))
{Search( Frames(i=0 to N-1), Input Page[k] )
  { at k,i:
    if (empty): then
      incr(Miss); Frames[i]= Input Page[k]
      New_mru(i)
    else {incr(Hit) and
      if (i != mru) then
        {if ( i = lru) then Interswitch-lru-mru()
        else Newstart_Rearrange(i) }
    }
  }
Not found (all non-empty) then
  { incr(Miss) and Replace(Frame[lru], Input Page[k])
  Interswitch-lru-mru() }
  
```

Basic Plan For OPT Main Program

```

Frames[1:N];
Input Pages[1:M];
MAX=N; LOC=0;
Read (Input Page (k=1 to M))
{ Search( Frames(i=1 to N), Input Page[k] )
  { at k, i:
    if (empty) then incr(Miss), Frames[i] =Input Page[k]
    if found: then incr(Hit) }
  }
Not found: (all non-empty) then
  incr(Miss) and MAX = k+1, LOC=0
  Read (Frames(i=1 to M))
  {Search(Input Page( j=k+1 to N),
    {at i, j at which InputPage (j) ==Frame(i),
    if j > MAX, MAX=j; LOC = i;}
  )
  }
  Replace(Frame[LOC], Input Page[MAX])
  
```

Procedures For LRU and OPT

toward_mru_link = bcklink
toward_lru_link=fwdlink

```

New_mru(i) /*moves mru pointer
to latest input and
bcklink(mru) = i; rearranges links*/
mru <- i
fwdlink[mru] <--temp;
bcklink[mru]<--fwdlink(temp)
  
```

Interswitch-lru-mru()

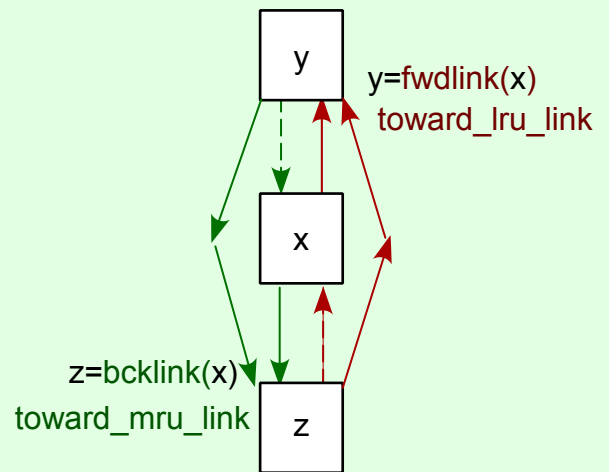
```

{ temp <-- {old}lru
  mru <-- {old}lru
  lru <-- bcklink[temp]
}
  
```

Newstart_Rearrange(x)

```

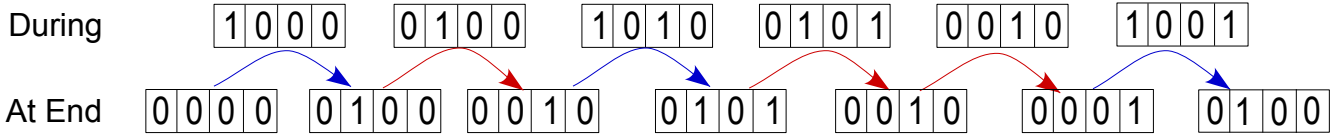
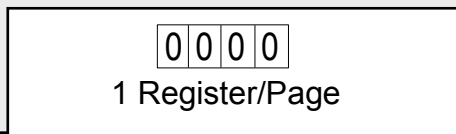
{ bcklink[fwdlink[x]] <-- bcklink[x]
  fwdlink[bcklink[x]] <-- fwdlink[x]
  fwdlink[x]<--{old}mru
  bcklink[{old}mru] <-- x
  fwdlink[{old}lru] <-- x
  bcklink[x] <-- {old}lru
  mru = x
}
  fwdlink[bcklink[x]] <-- fwdlink[x]
  fwdlink[bcklink[x]] <-- y
  fwdlink[z] <-- y
  
```



```

bcklink[fwdlink[x]] <-- bcklink[x]
bcklink[fwdlink[x]] <-- z
bcklink[y] <-- z
  
```

BASIC PLANS: OPT, FIFO (QUEUE, CIRCULAR), LRU



referenced during,
and shifted at end
of 1 Aging: clock period
2 Working Set References

not referenced during,
but shifted at end of
1 Aging: clock period
2 Working Set References

Time- Recent Use Those not recently used are candidates for replacement

AGING: Shift after Clock Period

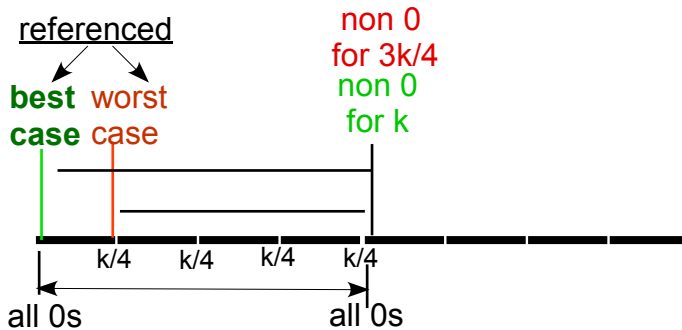
Register/Page--Replace Page with Lowest Number (binary) in Register. Tracks Both Recent and Frequent Use

Working Set = $w(k,t)$ = set of k most recently page references at time t .
For the right choice of k the resultant Working Set changes very slowly

WORKING SET Assuming $k = 4$: Shift after each Reference Any one having all 0s was *not* amongst the last 4 references. : If replacement is needed Replace any page not in Working Set

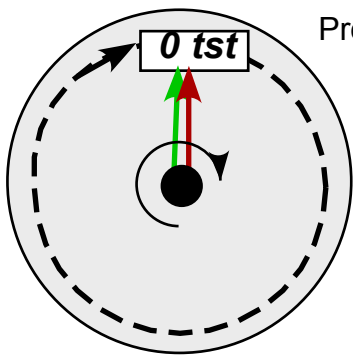
Working Set Approximation

If working set is large the registers per page would be prohibitively large. So suppose the working set were 20. The size 4 register could be used-but the shift would occur after 5 references.



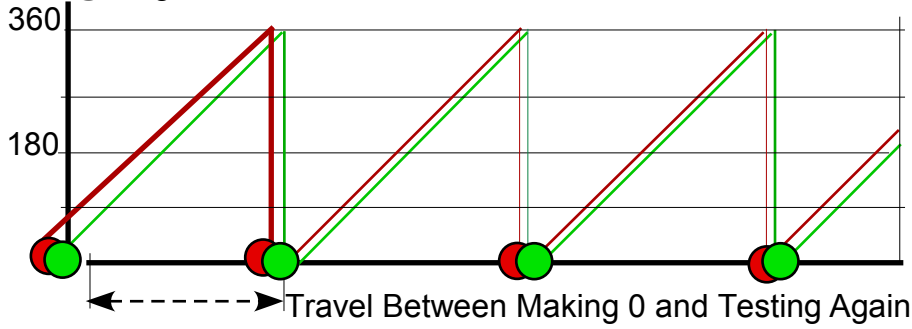
WORKING SET ALGORITHM APPROXIMATION :
Replace Page with all 0s- not amongst previous k references
(see page 12, 13 of these notes)

REPLACEMENT ALGORITHMS- AGING, WORKING SET (APPROX)



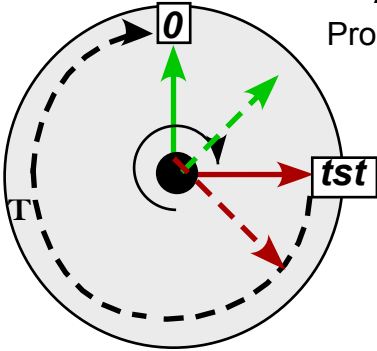
Effectively 1-Handed

Process @ Degree

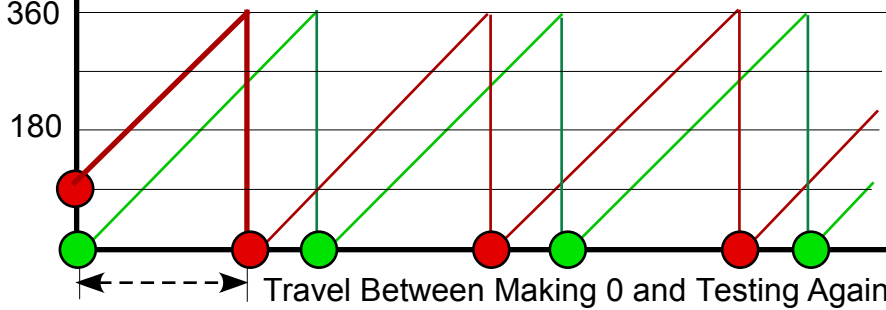


To make a replacement Travel Until Find a 0, replace-make it 1
All 1s passed in finding the 0 are made = 0

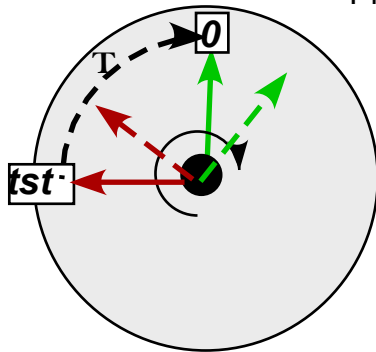
Replace the one which has been recorded as being present for the most time, unless it has been used in the *interval* determined by the distance between the two hands. If it has been used in that interval it not replaced. If it has not it is replaced.



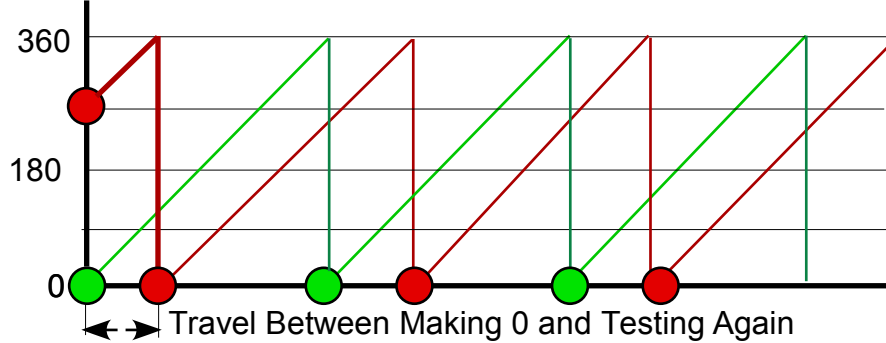
Process @ Degree



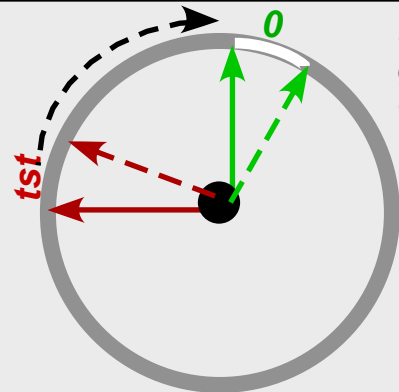
To make a replacement Travel Until RED Finds a 0, Replace- make 1
All 1s passed by GREEN In finding the 0 are made = 0



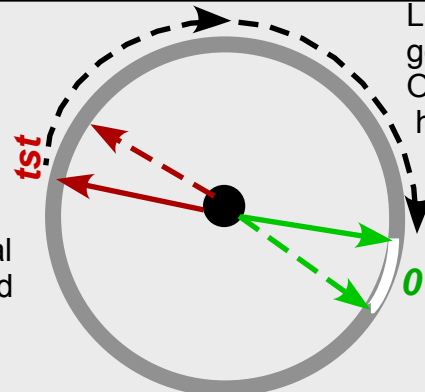
Process @ Degree



The time between the Green hand making a page position 0, and the RED hand reaching that page position is the time which the page has to become 1 due to being referenced. If it does so in that time then it will not be replaced when examined. By resetting the angle between hands that interval can be decreased if the Page Fault Frequency (PFF) increases or it can be increased if the PFF is low.



Short time to go from 0 to 1
OK if PFF is low.



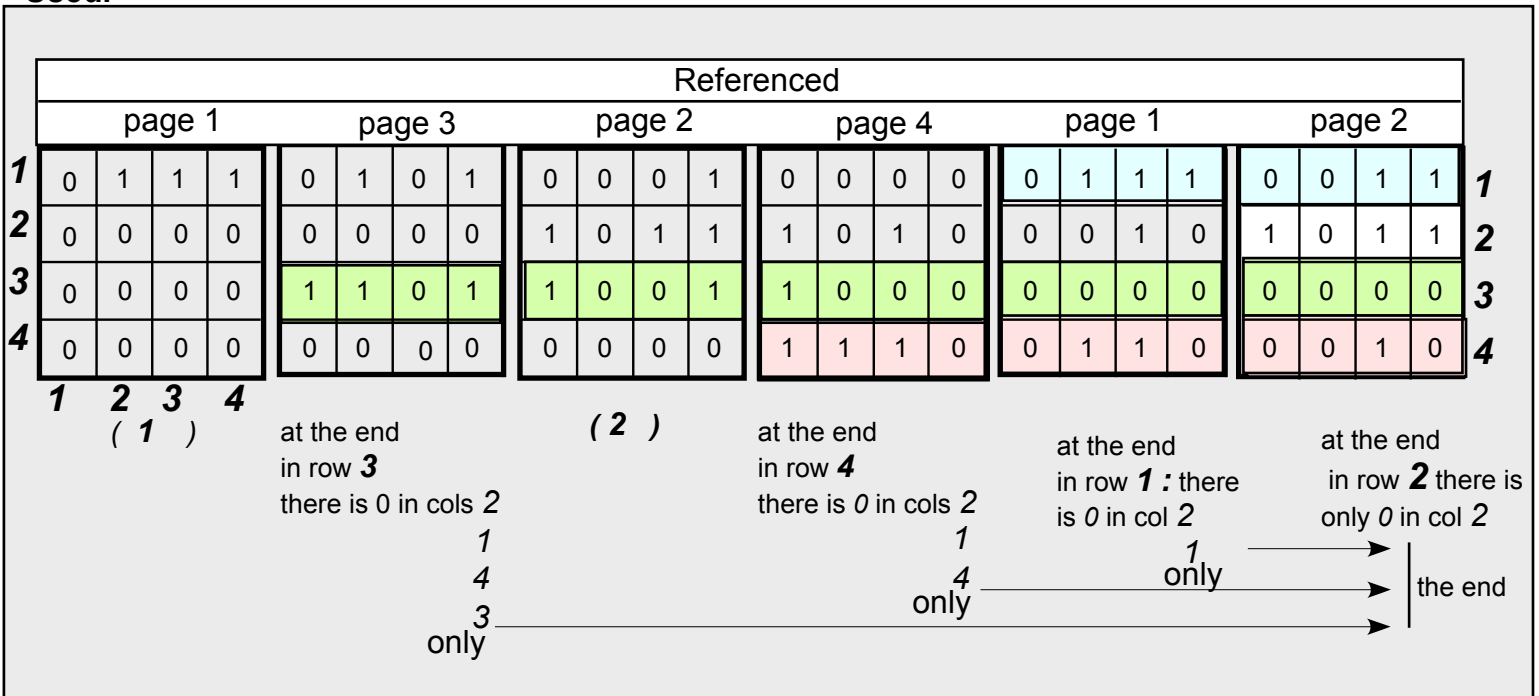
Longer time to go from 0 to 1
OK if PFF is higher

Only Make the search when Replacement or Removal to Tentative Free List is needed

- Test Hand- tests for 1 = recently referenced, if so do not replace, otherwise replace
- Clear Hand- make 0 = not recently referenced
- - - interval during which a page can be referenced and get a second chance

1-2 Handed Clock Algorithm Based On Second Chance-FIFO

The LRU Matrix has a distinct row and column for each page frame. Page frame i is assigned row i and column i . When **page i is referenced all entries in row i are made 1 and then all entries in column i are made 0.** At any time the **page(s) whose row has the most 0s is the Least Recently Used.**



In the final matrix there is exactly one 0 in row 2. There are exactly two 0s in row 1, because the last two pages referenced are 1 and 2. There are exactly three 0s in row 4 because the last three pages referenced were 1, 2 and 4. We can ignore the fact that another use of page 2 occurred earlier. There are exactly four 0s in row 3 because the last four pages referenced were 1, 2, 3, and 4.

After a row, R's last reference every other last reference puts a 0 in R's row.

if the last use of page P is followed by use of n other pages, different from P and from each other, there will be n 0s in P's row since each of the other pages will have put 0 in one of page P's columns. So the last used page, p_{last} , will have only one 0 in its row-the one which only has p_{last} (s) following it have 2 0s in its row, etc.

Let P_0 be the last page used and let P_1 be the page who last use is only followed by P_0 s and let P_2 be the page who last use is only followed by P_0 s and P_1 s and in general let P_{i+1} be the page whose last use is only followed P_0 and $P_1 \dots P_i$. Then at the end row P_0 has one 0s, and row P_1 has two 0s (in columns P_0 and P_1) P_i .has $i+1$ 0s (in rows P_0, P_1, \dots, P_i).

If the Aging algorithm (pg) were used and each Process were given a register with the number of bits = number of Processes the number of bits it would use the same number of bits as by this "Matrix" algorithm. The shift register in the Aging algorithm requires control hardware in addition to the storage. Writing 1's in rows and 0s in columns in the Matrix algorithm requires more time than just changing 1 bit when there is activity. The Matrix algorithm completely orders the recency of use of all Processes. The Aging algorithm approximates this same ordering (a good approximation if register length were the number of Processes). Either one is too costly and apparently their precision gives inadequate return.

REPLACEMENT ALGORITHMS TRUE LRU (More)- MATRIX IMPLEMENTATION

The Effectiveness of Clock Algorithms

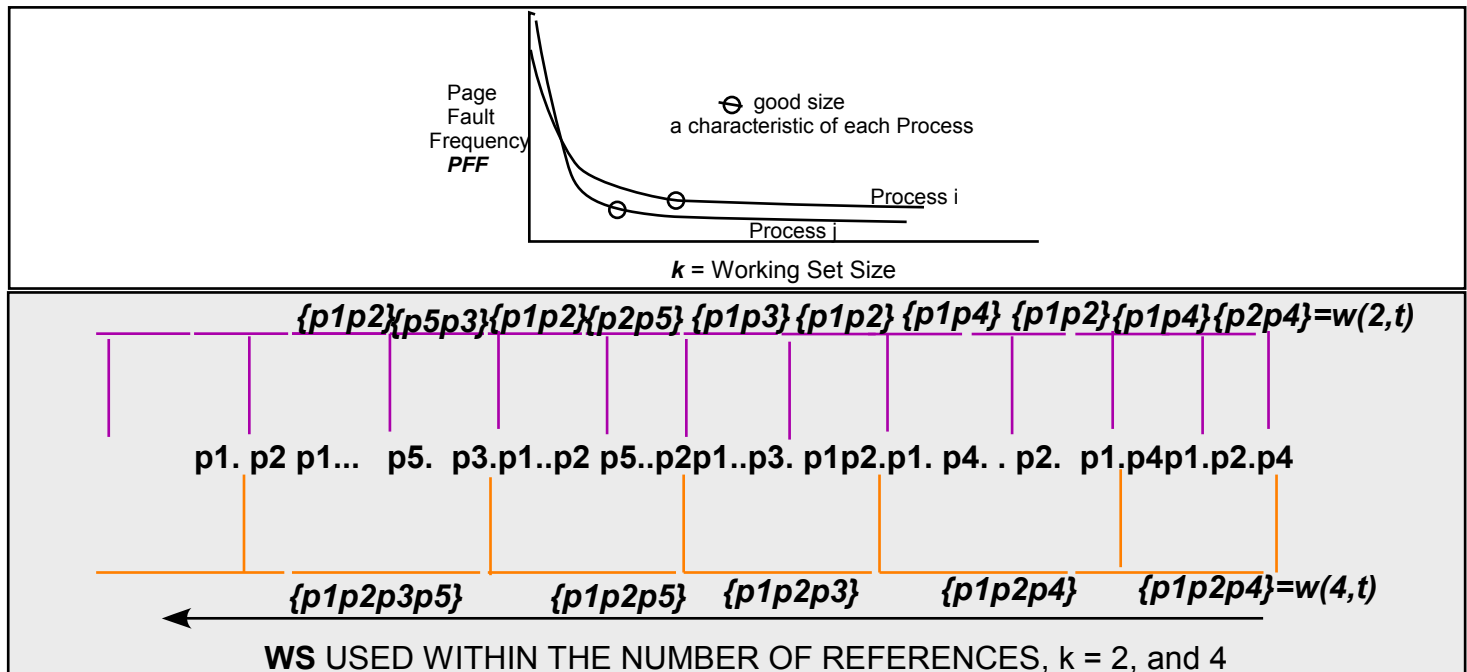
When a page enters the the Clock its reference becomes 0 and the clock hand moves clockwise to the next entry. So that a page will only be selected for possible replacement after the critical time. In this case the critical time is the time of one revolution of the Clock hand (pointer) = T_{return} . It can only be replaced if it has not been referenced for T_{return} . So the only pages that will be replaced are those which have not been referenced in T_{return} , and all pages referenced within a period T_{return} will be retained. If T_{return} is associated with the interval, I . T_{return} can vary, but can also be adjusted, based on the PFF. A second hand can be added. One hand reads the following one 0s and by adjusting the angle between the two hands the effective T_{return} can be varied as shown on page 7. Also one may selectively 0 bits of pages not used for a long time. In with. Note that T_{return} itself can be controlled also if one keeps a list of free pages and uses a daemon enabled periodically to replenish that list. This algorithm is an approximation to the Working Set algorithm approximations considered later-allowing adjustment of the working set size.

The Effectiveness of Aging Algorithms

Let T_{age} be the period between shifting the Aging registers. Aging with k bits will surely make all pages last referenced more than $k T_{age}$ intervals ago more susceptible to replacement than those that were..If all the pages were referenced within the last $k T_{age}$ intervals then all those which were last referenced, more than $k-1 T_{age}$ intervals ago will be more susceptible than all those that were.-etc. This is an approximation to LRU, closer to LRU than the working set algorithm considered next.

The Working Set Algorithm

Working Set = $w(k, t)$ = set of k most recently page references at time t . Replace any page not in working set (if possible)



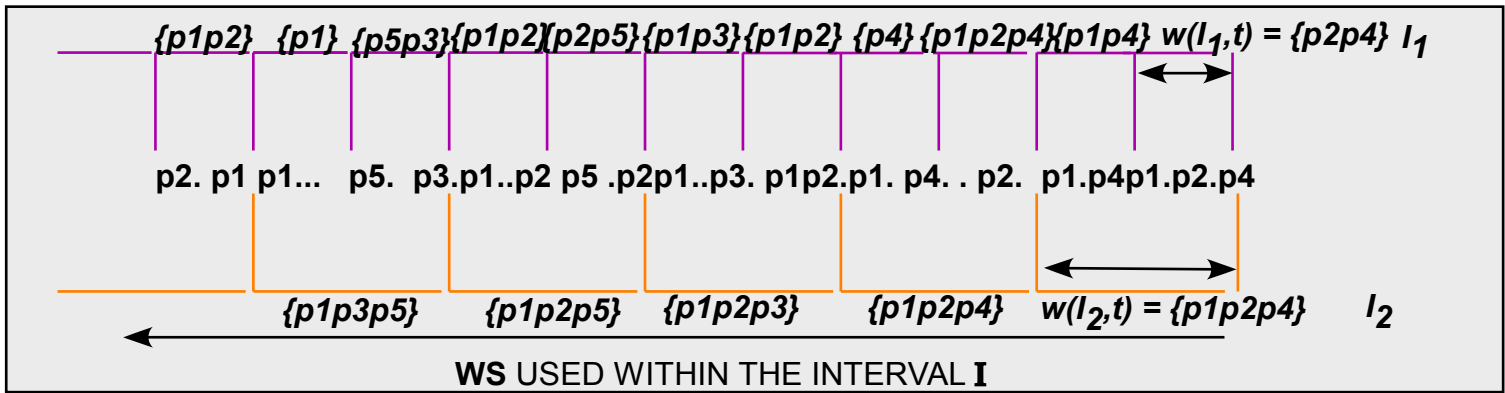
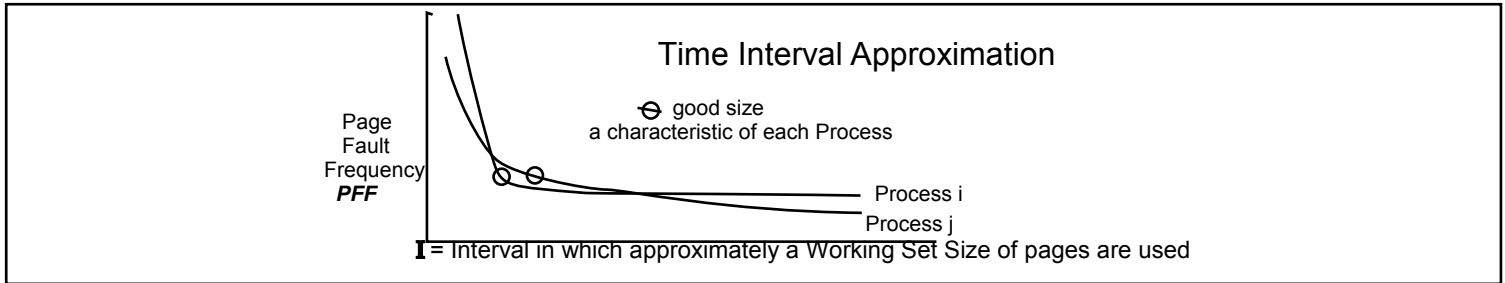
A straight forward way of keeping track of the working set is to record page references and the order in which they are made. This may be done by recording the first k reference in positions 1, 2, etc. of an array. After k references, each subsequent reference is made in the k th position after shifting the members in the array from j to $j-1$.

EFFECTIVENESS OF CLOCK AND AGING ALGORITHMS THE WORKING SET ALGORITHM

Working Set = $w(k, t)$ = set of k most recently page references at time t . Replace any page not in working set (if possible)

A simple working set replacement algorithm replaces any page in MM which is not in the working set. A more complex one could give a page outside the working set which is clean precedence for replacement over one outside the working set which is dirty.

Time Interval Approximation

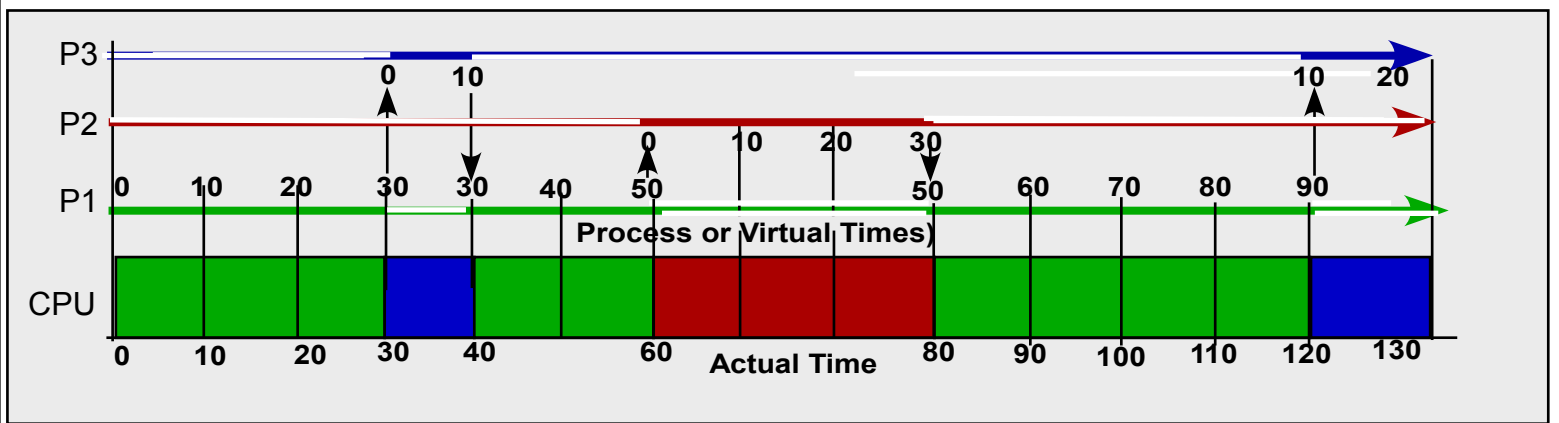


A simple way to test for activity in the interval I : 0 is placed in reference bit of pages at the beginning of each such interval. During the interval all pages referenced have their reference bit set to 1. At the end of the interval all pages with their reference bit = 1 are in the (approximate) working set. This allows us to know whether a reference bit was accessed in the last 0 through I time depending on when we have to do a page replacement. However this approach leaves us at the beginning of each interval with 0 pages referenced in the interval. There are better ways involving use of the Clock type of algorithm and recording the last time of reference of each page.

We first consider the details of the time interval approximation to the Working Set Algorithm, and Clock based approximations to it. Then we consider the algorithm which depends on the number of recent page references and its approximations.

Measuring Process Times

To obtain a useful working set algorithm: keep the *last virtual time* at which a process has been used. If it is *within an interval, I, before the current virtual time* for that Process it is in the working set and should not be replaced. If it was not used within that interval it may be replaced.



TIME INTERVAL APPROXIMATION WORKING SET ALGORITHM IN CLOCK ALGORITHM CONTEXT 1.

- If a page is referenced mark it 1--Choices:(a) Make all = 0 periodically(On each clock interrupt)
 (b) Only make 0 when 1 is detected during a replacement scan.
 (c) Make all 0 when there is a replacement.

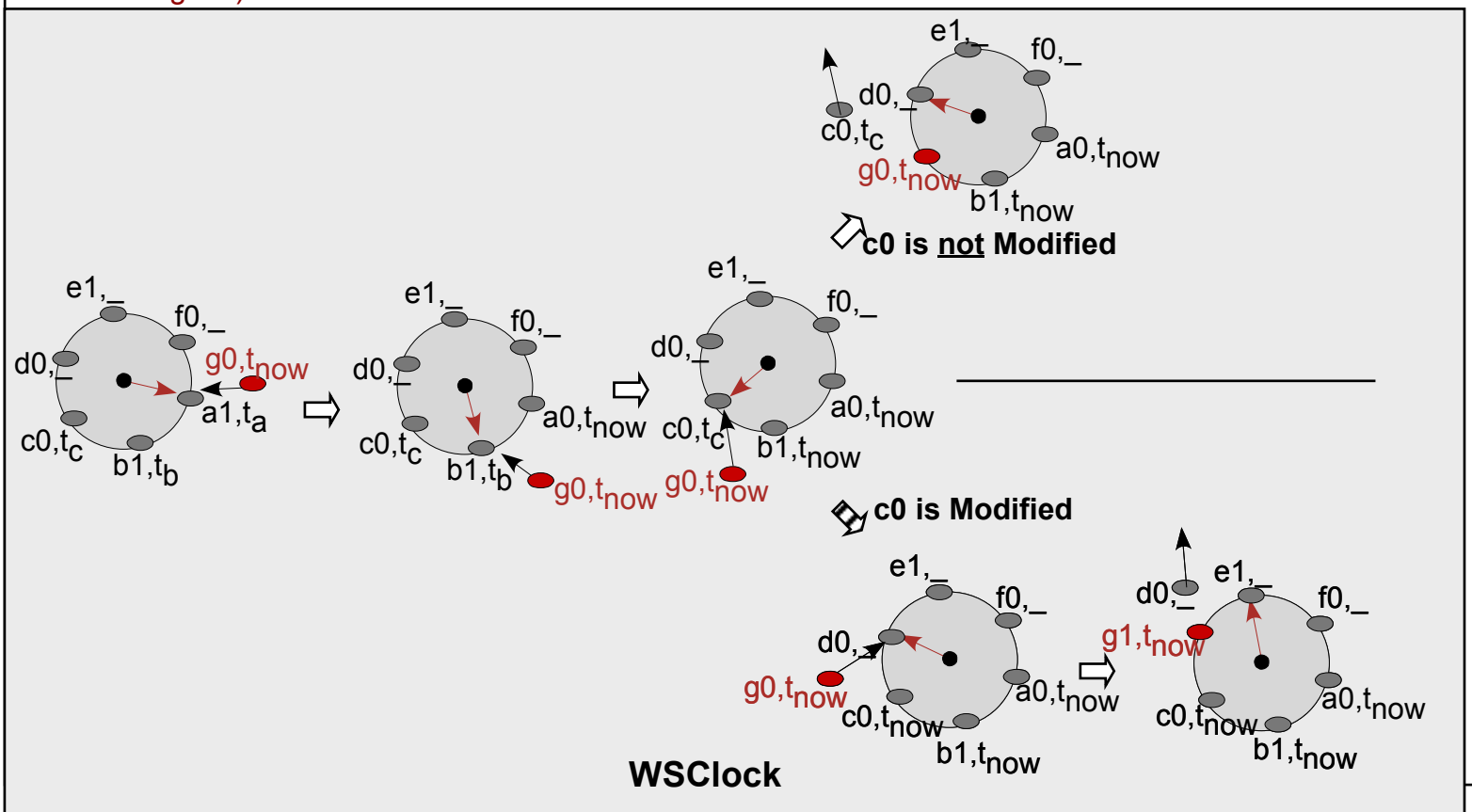
To use the time Interval, I , approximation, basically four items of information are needed. Namely:

- 1) The reference bit- R
- 2) The modified bit-M
- 3) The approximate time Pg was last referenced- T_{Iref} ,
- 4) Current time is t_{now} and

When there is a page fault, or Daemon looking for free page, thus a call for page replacement:

View pages in clock order,

- 1) if $R = 1$ **page is not replaced**, make $R = 0$, and put in (last referenced time), $T_{Iref} = t_{now}$, keep searching at pointer location
- 2) if $R = 0$ and
 - a) last referenced time $t_{nowt} - T_{Iref} < I$ **page is in WS and not replaced**, keep searching at next pointer location
 - b) last referenced time $\delta = t_{nowt} - T_{Iref} \geq I$ page is candidate for removal:
 - b1) If it has been modified and b1) not written back, if not already scheduled for writeback (The writeback is processed by a separate Process) it is so scheduled and not replaced, keep searching at next pointer location
 - b12) has been written back **page is replaced**, move pointer to next location and return
 - b2) If it has not been modified **page is replaced**, move pointer to next location and return
- c) It is possible that no page would have been chosen If any writes are scheduled can keep going. If none have been scheduled. Then can choose one for which $t_{nowt} - T_{Iref}$ (closest to being out of working set)



TIME INTERVAL APPROXIMATION WORKING SET ALGORITHM IN CLOCK ALGORITHM CONTEXT 2.

Current Time t_{now}	160		160+		162		163		166		167		...	170		170+	
	Fault I = 45													Fault I = 45			
P1	*100	0	160	0	160	1	160	0	160	1	160	0		160	1	170	0
P2	100	1	160	0	160	1	160	0	160	0	160	0		160	1	170	0
P3	75	1	160	0	160	0	160	0	160	1	160	0	...	160	0	160	0
P4	120	0	120	0	120	0	120	0	120	0	120	0		*120	0	170	0
P5	150	0	150	0	150	1	150	0	150	1	150	0		150	0	150	0
	T_{lref}	R	T_{lref}	R	T_{lref}	R	T_{lref}	R	T_{lref}	R	T_{lref}	R		T_{lref}	R	T_{lref}	R

WS-Modified Clock

ANOTHER EXAMPLE

This algorithm can be viewed as a modified LRU in which every page whose last use is prior to the k most recently used is equally available for replacement.

The Working Set Algorithms-Details

As an alternative to the interval approximation to the working set algorithm one can use:

a) One based directly on the working set, the number of page references n the working set window k at any time/. Then, for each process one keeps a size k pushdown list of the page identity of the page references so that the identity of the previous k page references for each process is always available.

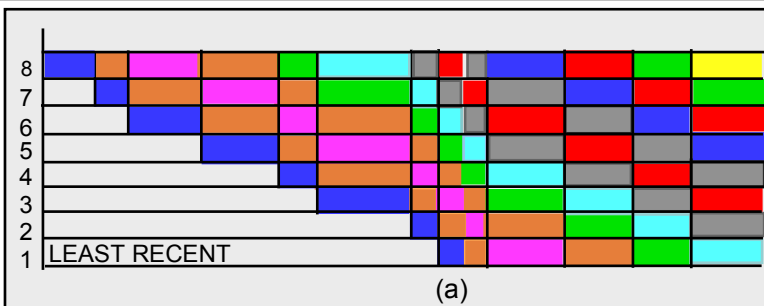
or

b) Another approach to the true working set algorithm, uses bitmaps. An array of k bits for each page is maintained. When the first page use occurs record 1 in the high order bit of the users array, When the next one is used shift all arrays right 1 place, and mark the high order bit of the second user 1. etc. Just before the $k+1$ st high order bit was set. The shift would remove the bit that was set when the first page use occurred.. From then on each use would remove the bit associated with that k uses earlier. This scheme would require many shifts but would always identify the k most recently used pages.

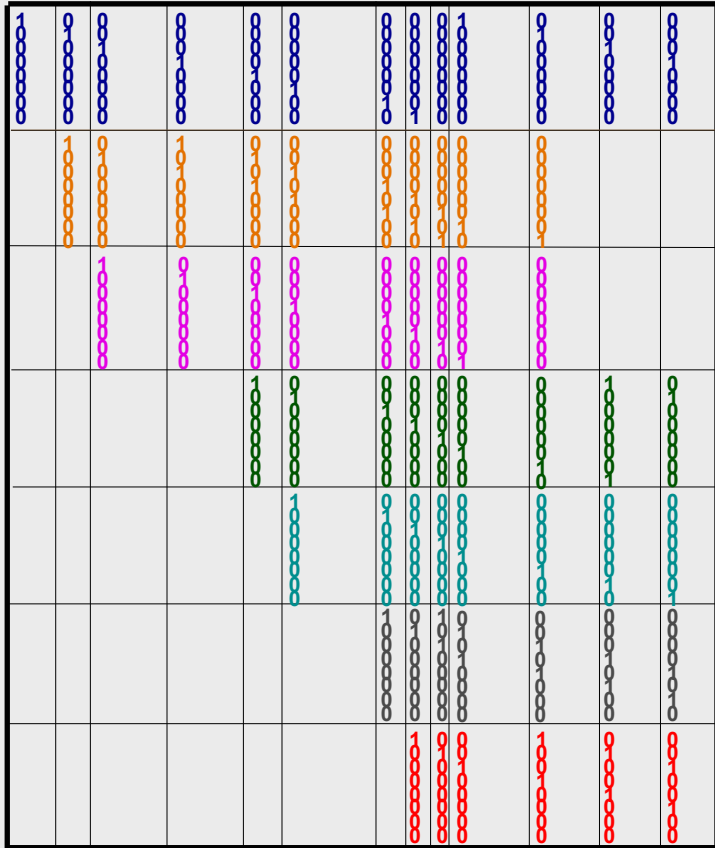
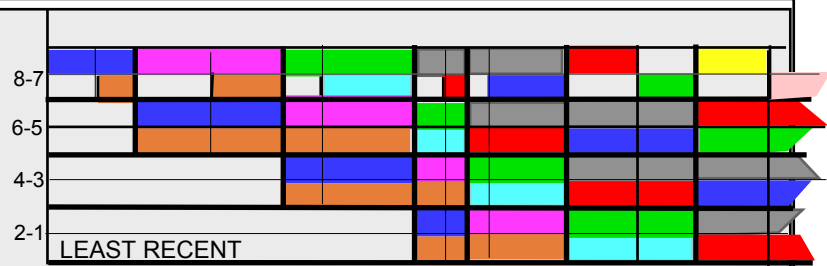
In an approximation version of this algorithm the shift would occur only after $k/n > 1$ page uses. After the k th use (k/n)th uses would be removed by the shift so only the k through the k/n most recent uses would be known. When the next k/n uses were recorded would again have a record of all k most recent uses.

The approach described in a) is illustrated in (a) of the following figure b. The different colors represent different pages. ($k = 8$). The top of the list is numbered 8. At all times after the first 8 pages have entered the list, the list shows the previous k page reference. If a page is to be removed it should not be one of these.

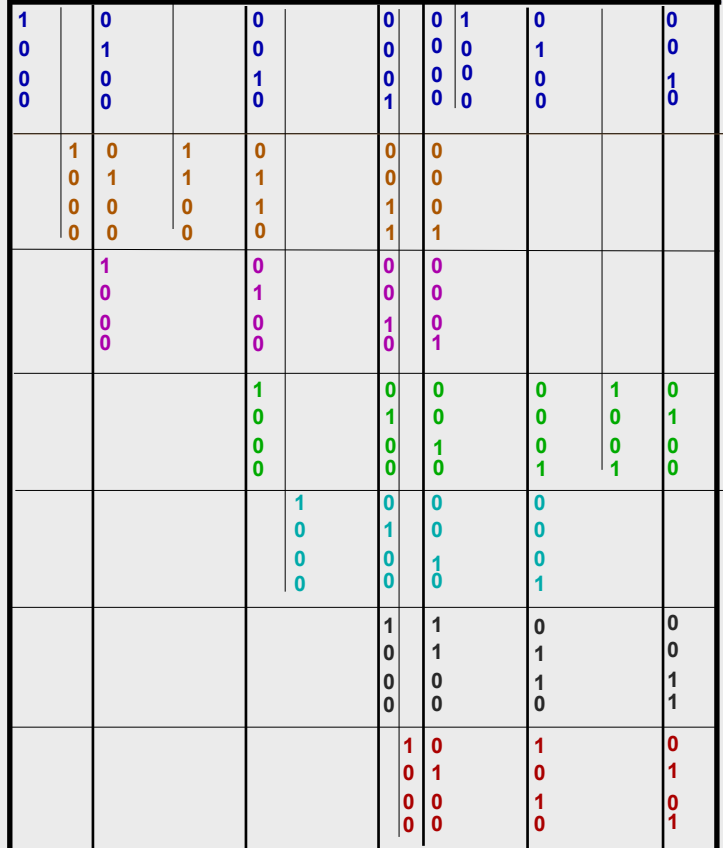
DIRECT APPROXIMATION WORKING SET ALGORITHM 1



(a)



(b)

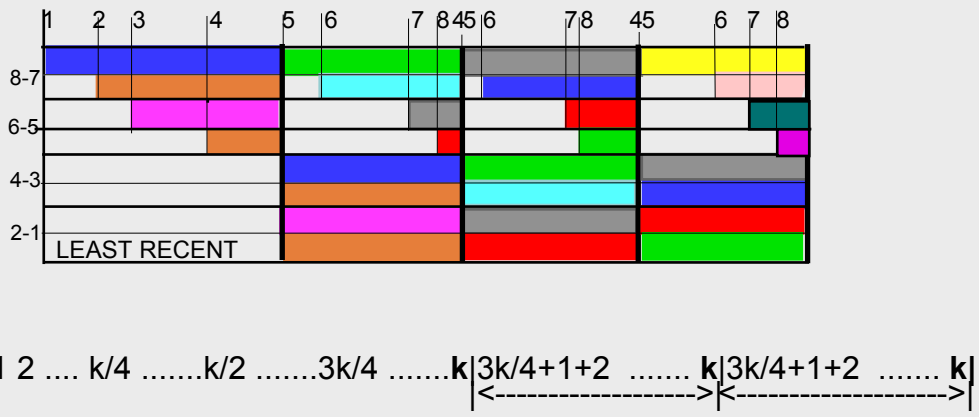


In (b) there is an equivalent set of pushdown “bit map” for recording this information. A bit map of size k is associated with each page. The top bit is marked 1 if that page is used, when the next, different, page is used the bit map for all pages are pushed down one position—the bit at the list bottom is discarded. (All unmarked positions are 0s). At any time then, only pages whose bit maps are all 0s are eligible for replacement

The approaches illustrated in (a) and (b) are probably too costly to be implemented. So one looks for an approximation.

The bit maps size can be cut in half if the push down of all maps are done after each successive pair of different references. If the same page is referenced in two successive pairs then there are a number of options. (Of course more space could be saved if the accumulation of page referenced registered between shifts were even greater). In our next example we have ignored such duplicate occurrences only if they occurred one after the other, In this example the same page input sequence illustrated as in the previous example, now however, the shifts only occur after every pair of page references. (Note that if the pages of the last k references were similarly grouped this would save the number of shifts, but not the number of page references which must be recorded)

BIT MAP APPROXIMATION WORKING SET ALGORITHM 2



The Effectiveness of the k/n Bit-map approximation to the Working Set Algorithm for w(k,t).

From the figure above it can be seen that the actual number of previous pages referenced at time t after the first k references will periodically vary between $((n-1)/n)k$ and k, changing at each page reference. Thus this algorithm approaches the ideal working set algorithm for w(k,t). as n decreases to 1.

The Replacement Algorithms Actually In Use

We have considered a fair number of possible algorithms for page replacement. The nagging question that has not been addressed is-which should be chosen? The efficiency in time and space is an important concern, as is the effectiveness for minimizing the number of page replacement is the other major concern.

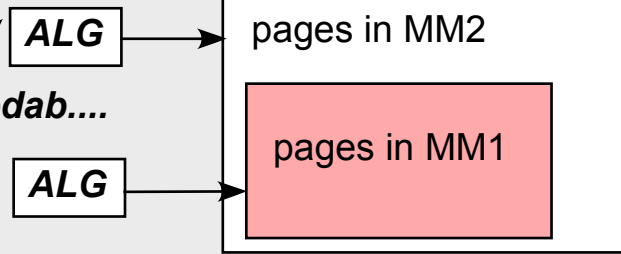
Apparently the algorithms actually in use all involve a clock. The major part of the program time is given to deciding on the page to be replaced when such a decision is called for. That is all the described algorithms are active both

- a) when a page is addressed-referenced, and/or modified. This occurs frequently. and
- b) when a search for a free(able) page is necessary which occurs much less frequently/

The Clock algorithms all involve a minimum of activity and space when a page is addressed. They reserve their major effort and space for implementation of the replacement decision is called for. On the otherhand, the Aging and Working Set algorithms based on page reference count need a bitmap of size > 1 for each page. The effectiveness grows with the number of bits). Aging approaches the perfect LRU algorithm, while the Working Set algorithm based on page reference approach the perfect Working Set Algorithm.

REPLACEMENT ALGORITHMS ACTUALLY USED

Page Sequence: *abcabcdedab...*

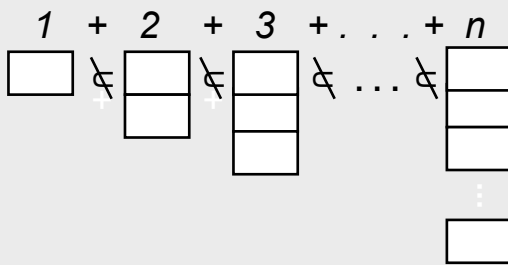


Given that MM1 has fewer page frames than MM2 that is $|MM1| < |MM2|$ and the same Replacement Algorithm, ALG is used on both, and the same Page Input Sequence is applied to both then:

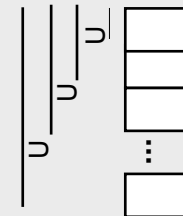
ALG is a STACK REPLACEMENT ALGORITHM. if:

- 1) MM2 requires no more Replacements than MM1 and
- 2) After each input every page in MM1 is also in MM2- Pages in MM1 \supseteq Pages MM2

Therefore:



For a Non-Stack Algorithm $1 + 2 + \dots + n = O(n^2)$ rows are required.



For a Stack Algorithm only n rows are required

		Page Input Sequence					
		a	b	c	a	d	
Page Frames	1	a	b	c	a	d	MM1 2 pages
	2	-*	a*	b*	c*	a	
		\cap	\cap	\cap	\cap	\cap	
	1	a	b	c	c	d	MM2 3 pages
	2	-*	a	b	b	c	
3	-	-*	a*	a*	b*		

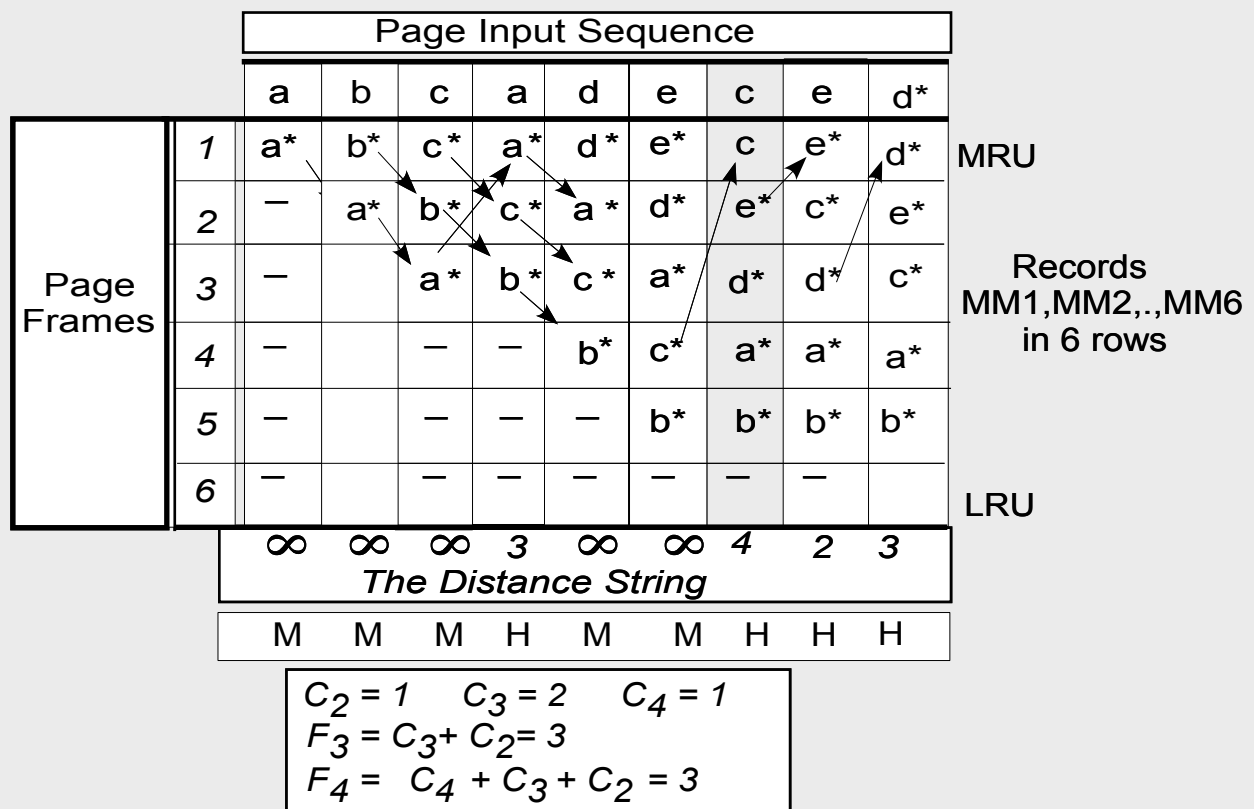
FIFO Replacement Algorithm

FIFO is not a STACK REPLACEMENT ALGORITHM.

So for FIFO it is necessary to keep tables having $1, 2, \dots, n$ rows to record the result of a simulation of all MMs having any number of pages from 1 to n .

So $1 + 2 + \dots + n = O(n^2)$ rows are required.

STACK REPLACEMENT ALGORITHM



C_i = the number of hits in row i , i.e. in the i th page of an i page frame memory.

F_i = the total number of hits in rows $j = 1$ to i i.e. in an i page frame memory.

$$\text{where } F_i = \sum_{j=1}^i C_j = C_1 + C_2 + \dots + C_i$$

To represent the result of LRU run on a given page input sequence:

- 1) if that page is not already in any row of the table, .put new page arrivals in row 1 and push all others down
- 2) If the incoming page, P, is already in a row of the table put P in row 1, and remove P's other occurrence from the row it was in, and finally push down all other entries filling in the blank left.

This is the same procedure one would use if there are a fixed number of pages, (therefore of rows,) except; when a new page came in and the MM was already filled the page at the bottom of table would be removed

**LRU IS A STACK REPLACEMENT ALGORITHM
In Which Tracking Hits In All Size MM < n Is Simple**

At any stage in the reception of the input page sequence the page that would be removed if the next page input required a replacement is the dispensable (*) one. The * in this representation has additional significance.

x^* in row k signifies that x is dispensable in a k page memory and, if the next row with a starred (*) entry is in row $k + i$, then x is dispensable also in row $k + 1$ to row $k + i - 1$. For LRU all entries are starred, so row n in a table with n rows (MM with n pages) is the dispensable page. This is not the case for OPT considered next.

*THE DISPENSABLE PAGE

STACK REPLACEMENT ALGORITHM- LRU IS ONE

The algorithm for filling in the table for OPT is not as simple as with LRU .

		a	b	c	a	d	e	c	e	d	b	a	e
1	a*	b*	c*	a*	d*	e*	c*	e*	d*	b*	a*	e*	
2	-	a	a	c	c	c	e	c*	e	e	e	a*	
3	-	-	b*	b	b*	d*	d	d	c*	d*	b*	b*	
4	-	-	-	-	a*	b*	b	b	b	c*	d*	d*	
5	-	-	-	-	-	a*	a	a	a	a	c*	c*	
6	-	-	-	-	-	-	-	-	-	-			
		∞	∞	∞	2	∞	∞	2	2	3	4	5	2

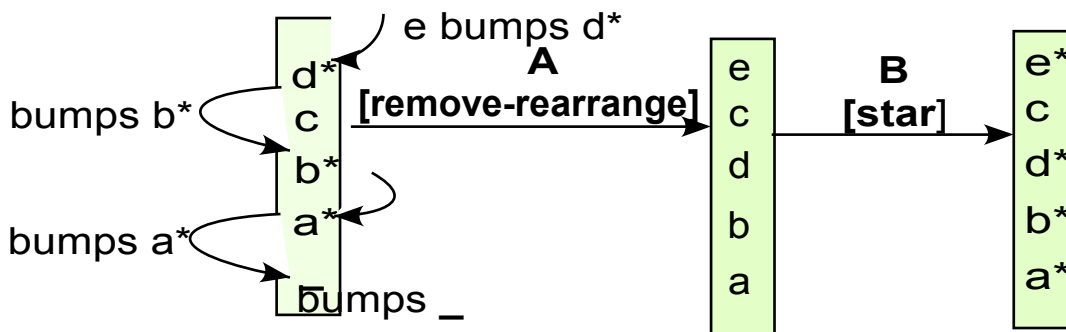
$$C_2 = 4 \quad C_3 = 1 \quad C_4 = 1 \quad C_5 = 1$$

$$F_3 = C_3 (=1) + C_2 (=3) = 4$$

$$F_4 = C_3 (=1) + C_3 (=1) + C_4 (=1) = 5$$

LRU IS A STACK REPLACEMENT ALGORITHM
In Which Tracking Hits In All Size MM < n Is Not Less Simple Than For LRU

Unlike LRU, in which all pages in the table are starred, When a new entry is made in OPT, it is moved into the size 1 MM page, and made dispensible there. For determining its effect on the content of other size pages in MM. the “bumping” algorithm illustrated below for column 6 of the above example is used. To determine which pages are dispensible in other pages the basic OPT algorithm, i.e whose next reference is furthest is used.



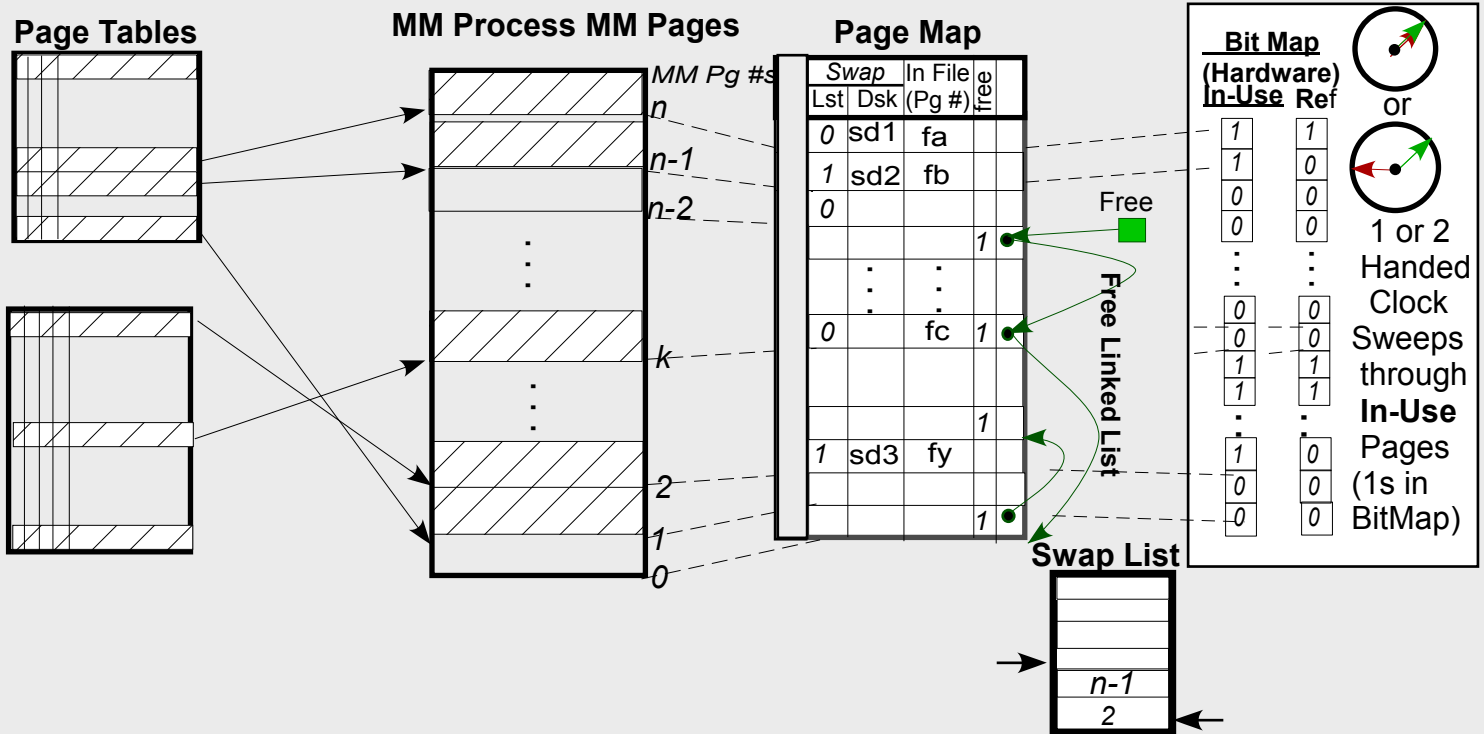
- A [remove-rearrange]** The dispensible pages in each column are determined by the order in which those pages in MM will again arrive in the future.
- B [star]** The page in a 1 page MM is always starred, say it is x*. Then, moving to larger and larger size MMs one looks for a page that is further in the input sequence if and when that is found, one stars it say y*. One continues examining larger MMs until one further than y* is found and so on.

OPT - A STACK REPLACEMENT ALGORITHM

Daemons: Independent OS Processes that are awakened when the CPU is available or PFF is low but sufficiently often to perform their job at a reasonable rate.

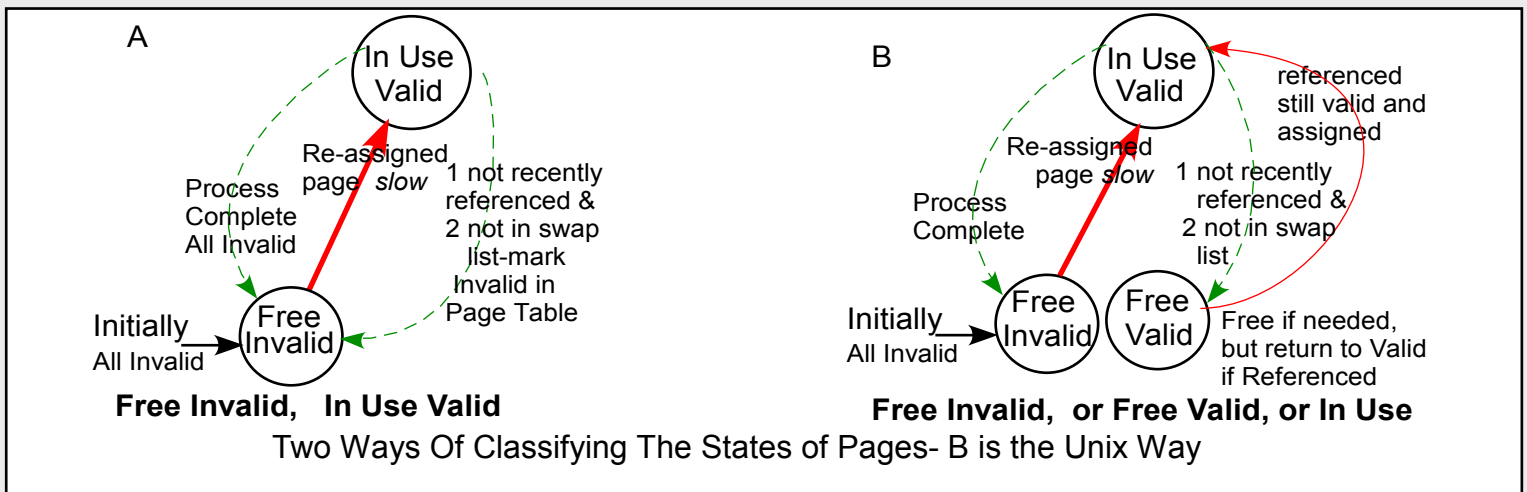
Examples:

1. **Daemons:** Associated With Paging-Updating Page Free List



Contains MM Pg #s. Pages to be written out. not free even if seldom used before then.

When a page is assigned to a Process it is marked **In-Use** in Page Map. When a page is referenced it is marked 1 in **Ref** in Page Map. When more pages are to be put on Free List a page with **In-Use = 1** & **Ref = 0** which is not on Swap List is eligible. When a page is freed it becomes invalid-**In-Use** -> 0 this information is either sent to Process Table or Process must check **In-Use** associated with Page Map when that page is referenced by Process



Two Ways Of Classifying The States of Pages- B is the Unix Way

In B Keep an adequate supply of Free Pages (if PFF is High Need More). Replacements can be done when there is time. It is necessary also to manage the Swap List

When a Page is modified it is put in the **Swap List** and Marked 1 in the Page Map in column **Swap Lst**. When it leaves the **Swap List** it is marked 0 in Page Map in column **Swap Lst**

A page that hasn't been recently referenced is put on **Free (Link) List** (Page Map) if it is not 1 in **Swap List**. A page on the **Free List** which is referenced and still valid is removed from the **Free List** made **In-Use**.

UNIX DEMAND PAGING: DAEMONS, SWAP LIST, PAGE MAP

Daemons: (pg 691) Background Independent OS Processes that are awakened when the CPU is available or PFF is low but sufficiently often to perform their job at a reasonable rate

Examples:

2) cron daemon. Wakes every minute does necessary chores: like:

(a) reminder notices (a beep for appointments)

(b) daily disk backups

3) Manage Printer Queue

4) Incoming and outgoing mail

DAEMONS, CRON, PRINTER, MAIL