



Page Replacement Algorithms

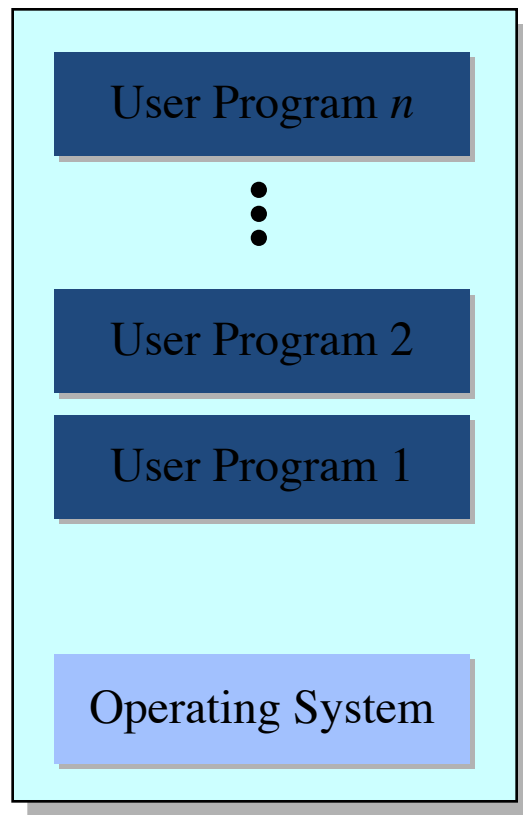
Don Porter

Portions courtesy Emmett Witchel and Kevin Jeffay



Virtual Memory Management: Recap

- Key concept: Demand paging
 - Load pages into memory only when a page fault occurs
- Issues:
 - Placement strategies
 - Place pages anywhere – no placement policy required
 - Replacement strategies
 - What to do when there exist more jobs than can fit in memory
 - Load control strategies
 - Determining how many jobs can be in memory at one time



Memory



Page Replacement Algorithms

- Typically $\sum_i VAS_i \gg \text{Physical Memory}$
- With demand paging, physical memory fills quickly
- When a process faults & memory is full, some page must be swapped out
 - Handling a page fault now requires **2** disk accesses not 1!

Which page should be replaced?

Local replacement — Replace a page of the faulting process

Global replacement — Possibly replace the page of another process



Page Replacement: Eval. Methodology

- Record a *trace* of the pages accessed by a process
 - Example: (Virtual page, offset) address trace...
(3,0), (1,9), (4,1), (2,1), (5,3), (2,0), (1,9), (2,4), (3,1), (4,8)
 - generates page trace
3, 1, 4, 2, 5, 2, 1, 2, 3, 4 (represented as *c, a, d, b, e, b, a, b, c, d*)
- Hardware can tell OS when a new page is loaded into the TLB
 - Set a used bit in the page table entry
 - Increment or shift a register

Simulate the behavior of a page replacement algorithm on the trace and record the number of page faults generated

fewer faults → *better performance*



Optimal Strategy: Clairvoyant Replacement

- Replace the page that won't be needed for the longest time in the future

Initial allocation

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>									
1	<i>b</i>										
2	<i>c</i>										
3	<i>d</i>										
Faults											
Time page needed next											





Optimal Strategy: Clairvoyant Replacement

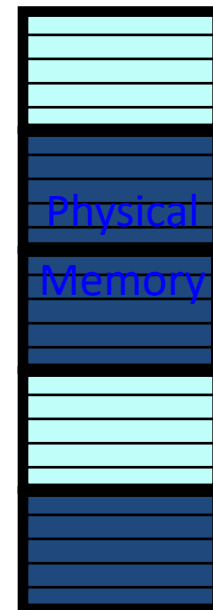
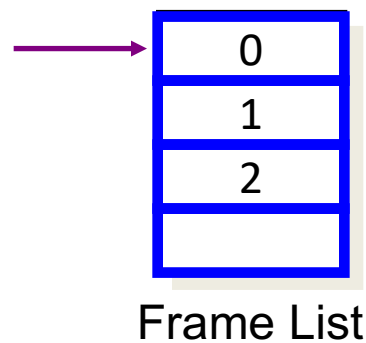
- Replace the page that won't be needed for the longest time in the future

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>d</i>
	1	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>
	2	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>
	3	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>
Faults						•					•
Time page needed next					<i>a</i> = 7 <i>b</i> = 6 <i>c</i> = 9 <i>d</i> = 10					<i>a</i> = 15 <i>b</i> = 11 <i>c</i> = 13 <i>d</i> = 14	



Local Replacement: FIFO

- Simple to implement
 - A single pointer suffices
- Performance with 4 page frames:



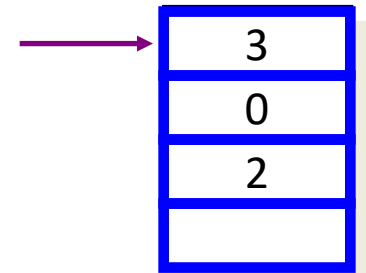
Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames		<i>a</i>									
0		<i>a</i>									
1		<i>b</i>									
2		<i>c</i>									
3		<i>d</i>									
Faults											



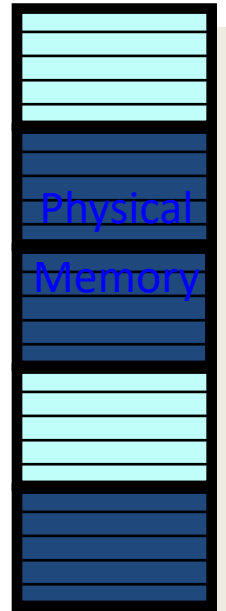


Local Replacment: FIFO

- Simple to implement
 - A single pointer suffices
- Performance with 4 page frames:



Frame List



Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>d</i>
	1	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
	2	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>b</i>	<i>b</i>	<i>b</i>
	3	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>c</i>	<i>c</i>
Faults						•		•	•	•	•



Least Recently Used (LRU) Replacement

- Use the recent past as a predictor of the near future
- Replace the page that hasn't been referenced for the longest time

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>										
	1	<i>b</i>										
	2	<i>c</i>										
	3	<i>d</i>										
Faults												
Time page last used												





Least Recently Used (LRU) Replacement

- Use the recent past as a predictor of the near future
- Replace the page that hasn't been referenced for the longest time

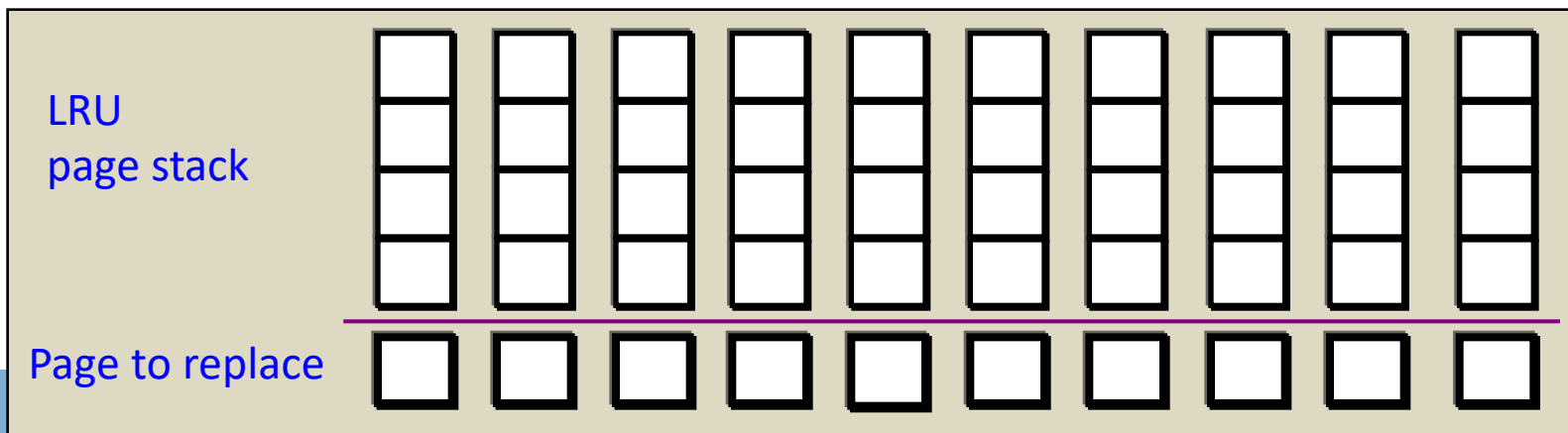
Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
	1	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>
	2	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>d</i>
	3	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>c</i>	<i>c</i>
Faults						•				•	•
Time page last used					<i>a</i> = 2 <i>b</i> = 4 <i>c</i> = 1 <i>d</i> = 3				<i>a</i> = 7 <i>b</i> = 8 <i>e</i> = 5 <i>d</i> = 3	<i>a</i> = 7 <i>b</i> = 8 <i>e</i> = 5 <i>c</i> = 9	



How to Implement LRU?

- Maintain a “stack” of recently used pages

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a
	1	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	d
	3	d	d	d	d	d	d	d	d	c	c
Faults						•				•	•

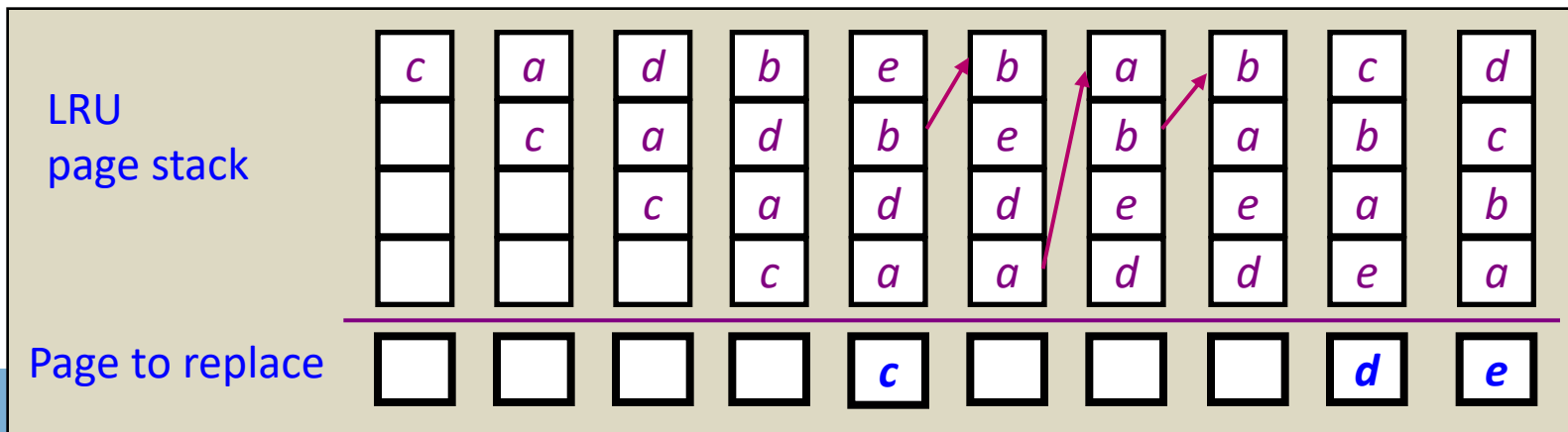




How to Implement LRU?

- Maintain a “stack” of recently used pages

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a
	1	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	d
	3	d	d	d	d	d	d	d	d	c	c
Faults						•				•	•



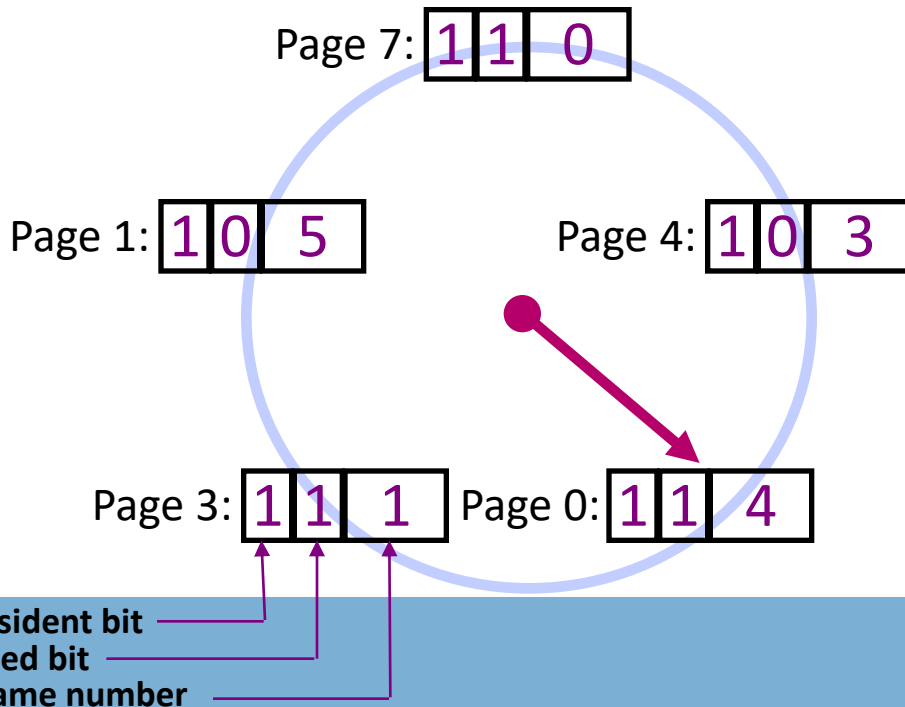


- What is the goal of a page replacement algorithm?
 - A. Make life easier for OS implementer
 - B. Reduce the number of page faults
 - C. Reduce the penalty for page faults when they occur
 - D. Minimize CPU time of algorithm



Approximate LRU: The Clock Algorithm

- Maintain a circular list of pages resident in memory
 - Use a *clock* (or *used/referenced*) bit to track how often a page is accessed
 - The bit is set whenever a page is referenced
- Clock hand sweeps over pages looking for one with *used* bit = 0
 - Replace pages that haven't been referenced for one complete revolution of the clock



```
func Clock_Replacement
begin
  while (victim page not found) do
    if (used bit for current page = 0) then
      replace current page
    else
      reset used bit
    end if
    advance clock pointer
  end while
end Clock_Replacement
```



Clock Example

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>						
	1	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>						
	2	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>						
	3	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>						
Faults											

Page table entries
for resident pages:

1	<i>a</i>
1	<i>b</i>
1	<i>c</i>
1	<i>d</i>





Clock Example

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>d</i>
	1	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>
	2	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
	3	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>c</i>	<i>c</i>
Faults						•		•		•	•

Page table entries
for resident pages:

1	<i>a</i>
1	<i>b</i>
1	<i>c</i>
1	<i>d</i>

1	<i>e</i>
0	<i>b</i>
0	<i>c</i>
0	<i>d</i>

1	<i>e</i>
1	<i>b</i>
0	<i>c</i>
0	<i>d</i>

1	<i>e</i>
1	<i>b</i>
1	<i>a</i>
0	<i>d</i>

1	<i>e</i>
1	<i>b</i>
1	<i>a</i>
0	<i>d</i>

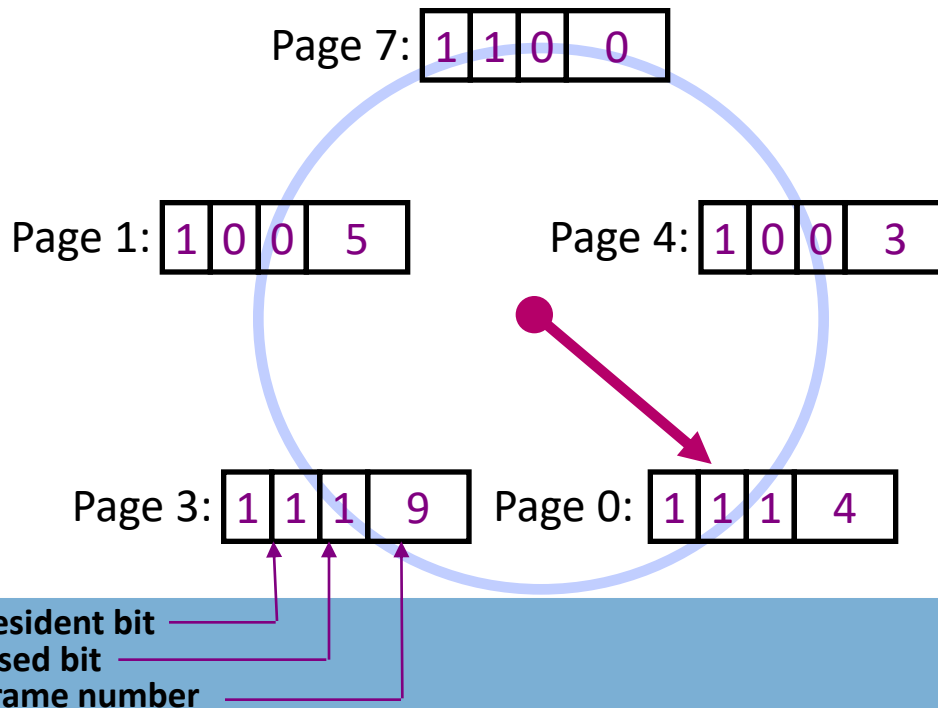
1	<i>e</i>
1	<i>b</i>
1	<i>a</i>
1	<i>c</i>

1	<i>d</i>
0	<i>b</i>
0	<i>a</i>
0	<i>c</i>



Optimization: Second Chance Algorithm

- There is a significant cost to replacing “dirty” pages
 - Why?
 - Must write back contents to disk before freeing!
- Modify the Clock algorithm to allow dirty pages to always survive one sweep of the clock hand
 - Use both the *dirty bit* and the *used bit* to drive replacement



Second Chance Algorithm

Before clock sweep		After clock sweep	
<i>used</i>	<i>dirty</i>	<i>used</i>	<i>dirty</i>
0	0		
0	1	0	0
1	0	0	0
1	1	0	1

replace page



Second Chance Example

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a^w</i>	<i>d</i>	<i>b^w</i>	<i>e</i>	<i>b</i>	<i>a^w</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>						
	1	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>						
	2	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>						
	3	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>						
Faults											

Page table
entries
for resident
pages:

10	<i>a</i>
10	<i>b</i>
10	<i>c</i>
10	<i>d</i>





Second Chance Example

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>a^w</i>	<i>d</i>	<i>b^w</i>	<i>e</i>	<i>b</i>	<i>a^w</i>	<i>b</i>	<i>c</i>	<i>d</i>
Page Frames	0	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
	1	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>d</i>
	2	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>
	3	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>c</i>	<i>c</i>
Faults						•				•	•

Page table
entries for
resident
pages:

10	<i>a</i>
10	<i>b</i>
10	<i>c</i>
10	<i>d</i>

11	<i>a</i>
11	<i>b</i>
10	<i>c</i>
10	<i>d</i>

00	<i>a</i> *
00	<i>b</i> *
10	<i>e</i>
00	<i>d</i>

00	<i>a</i>
10	<i>b</i>
10	<i>e</i>
00	<i>d</i>

11	<i>a</i>
10	<i>b</i>
10	<i>e</i>
00	<i>d</i>

11	<i>a</i>
10	<i>b</i>
10	<i>e</i>
10	<i>c</i>

00	<i>a</i> *
10	<i>d</i>
00	<i>e</i>
00	<i>c</i>



Page Replacement Performance

- Local page replacement
 - LRU — Ages pages based on when they were last used
 - FIFO — Ages pages based on when they're brought into memory
- Towards global page replacement ... with variable number of page frames allocated to processes

The principle of locality

- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions
- When processing large data structures, the dominant cost is sequential processing on individual structure elements
- Temporal vs. physical locality



Optimal Replacement with a Variable Number of Frames

- *VMIN* — Replace a page that is not referenced in the *next* τ accesses
- Example: $\tau = 4$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>c</i>	<i>e</i>	<i>a</i>	<i>d</i>
Pages in Memory	Page <i>a</i>	• <i>t = 0</i>									
	Page <i>b</i>	-									
	Page <i>c</i>	-									
	Page <i>d</i>	• <i>t = -1</i>									
	Page <i>e</i>	-									
Faults											





Optimal Replacement with a Variable Number of Frames

- *VMIN* — Replace a page that is not referenced in the *next* τ accesses
- Example: $\tau = 4$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>c</i>	<i>e</i>	<i>a</i>	<i>d</i>
Pages in Memory	Page <i>a</i>	• <i>t=0</i>	-	-	-	-	-	-	-	<i>F</i>	-
	Page <i>b</i>	-	-	-	<i>F</i>	-	-	-	-	-	-
	Page <i>c</i>	-	<i>F</i>	•	•	•	•	•	-	-	-
	Page <i>d</i>	• <i>t=-1</i>	•	•	•	-	-	-	-	-	<i>F</i>
	Page <i>e</i>	-	-	-	-	-	<i>F</i>	•	•	-	-
Faults		•			•		•			•	•



The Working Set Model

- Assume recently referenced pages are likely to be referenced again soon...
- ... and *only* keep those pages recently referenced in memory (called *the working set*)
 - Thus pages may be removed even when no page fault occurs
 - The number of frames allocated to a process will vary over time
- A process is allowed to execute only if its working set fits into memory
 - The working set model performs implicit load control



Working Set Page Replacement

- Keep track of the last τ references (including faulting reference)
 - The pages referenced during the last τ memory accesses are the working set
 - τ is called the *window size*
- Example: Working set computation, $\tau = 4$ references:

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>c</i>	<i>e</i>	<i>a</i>	<i>d</i>
Pages in Memory	Page <i>a</i>	• $t = 0$									
	Page <i>b</i>	-									
	Page <i>c</i>	-									
	Page <i>d</i>	• $t = -1$									
	Page <i>e</i>	• $t = -2$									
Faults											





Working Set Page Replacement

- Keep track of the last τ references
 - The pages referenced during the last τ memory accesses are the working set
 - τ is called the *window size*
- Example: Working set computation, $\tau = 4$ references:

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>c</i>	<i>e</i>	<i>a</i>	<i>d</i>
Pages in Memory	Page <i>a</i>	$t=0$ •	•	•	-	-	-	-	-	<i>F</i>	•
	Page <i>b</i>	-	-	-	<i>F</i>	•	•	•	-	-	-
	Page <i>c</i>	-	<i>F</i>	•	•	•	•	•	•	•	•
	Page <i>d</i>	$t=-1$ •	•	•	•	•	•	-	-	-	<i>F</i>
	Page <i>e</i>	$t=-2$ •	•	-	-	-	-	<i>F</i>	•	•	•
Faults		•			•		•			•	•



Page-Fault-Frequency Page Replacement

- An alternate approach to computing working set
- Explicitly attempt to minimize page faults
 - When page fault frequency is high — *increase working set*
 - When page fault frequency is low — *decrease working set*

Algorithm:

Keep track of the rate at which faults occur

When a fault occurs, compute the time since the last page fault

Record the time, t_{last} , of the last page fault

If the time between page faults is “large” then reduce the working set

If $t_{current} - t_{last} > \tau$, then remove from memory all pages not referenced in $[t_{last}, t_{current}]$

If the time between page faults is “small” then increase working set

If $t_{current} - t_{last} \leq \tau$, then add faulting page to the working set



Page Fault Frequency Replacement

- Example, window size = 2
- If $t_{current} - t_{last} > 2$, remove pages not referenced in $[t_{last}, t_{current}]$ from the working set
- If $t_{current} - t_{last} \leq 2$, just add faulting page to the working set

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>c</i>	<i>e</i>	<i>a</i>	<i>d</i>
Pages in Memory	Page <i>a</i>	•									
	Page <i>b</i>	-									
	Page <i>c</i>	-									
	Page <i>d</i>	•									
	Page <i>e</i>	•									
Faults											
$t_{cur} - t_{last}$											





Page Fault Frequency Replacement

- Example, window size = 2
- If $t_{current} - t_{last} > 2$, remove pages not referenced in $[t_{last}, t_{current}]$ from the working set
- If $t_{current} - t_{last} \leq 2$, just add faulting page to the working set

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		<i>c</i>	<i>c</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>c</i>	<i>e</i>	<i>a</i>	<i>d</i>
Pages in Memory	Page <i>a</i>	•	•	•	-	-	-	-	-	<i>F</i>	•
	Page <i>b</i>	-	-	-	<i>F</i>	•	•	•	•	-	-
	Page <i>c</i>	-	<i>F</i>	•	•	•	•	•	•	•	•
	Page <i>d</i>	•	•	•	•	•	•	•	•	-	<i>F</i>
	Page <i>e</i>	•	•	•	•	-	-	<i>F</i>	•	•	•
Faults		•			•		•			•	•
$t_{cur} - t_{last}$		1			3		2			3	1



Load Control: Fundamental Trade-off

- High multiprogramming level

$$\text{➤ } MPL_{max} = \frac{\textit{number of page frames}}{\textit{minimum number of frames required for a process to execute}}$$

- ◆ Low paging overhead

$$\text{➤ } MPL_{min} = 1 \text{ process}$$

- ◆ Issues

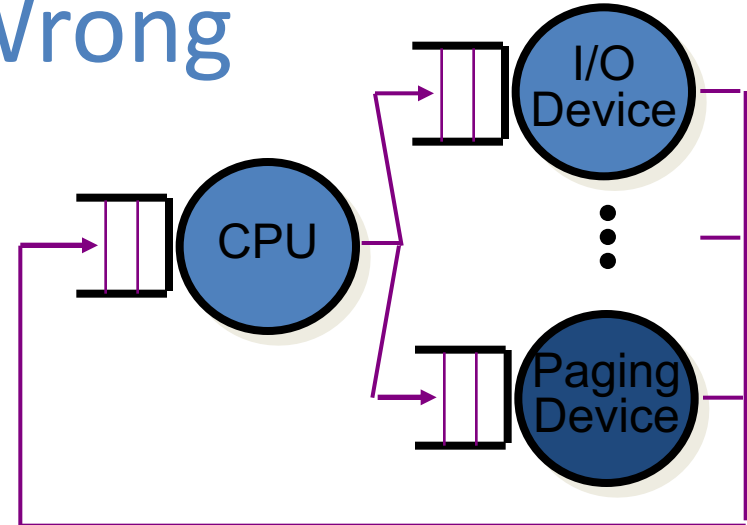
- What criterion should be used to determine when to increase or decrease the MPL ?
- Which task should be swapped out if the MPL must be reduced?



Load Control Done Wrong

i.e., based on CPU utilization

- ◆ Assume memory is nearly full
- ◆ A chain of page faults occur
 - A queue of processes forms at the paging device
- ◆ CPU utilization falls
- Operating system increases *MPL*
 - New processes fault, taking memory away from existing processes
- CPU utilization goes to 0, the OS increases the *MPL* further...

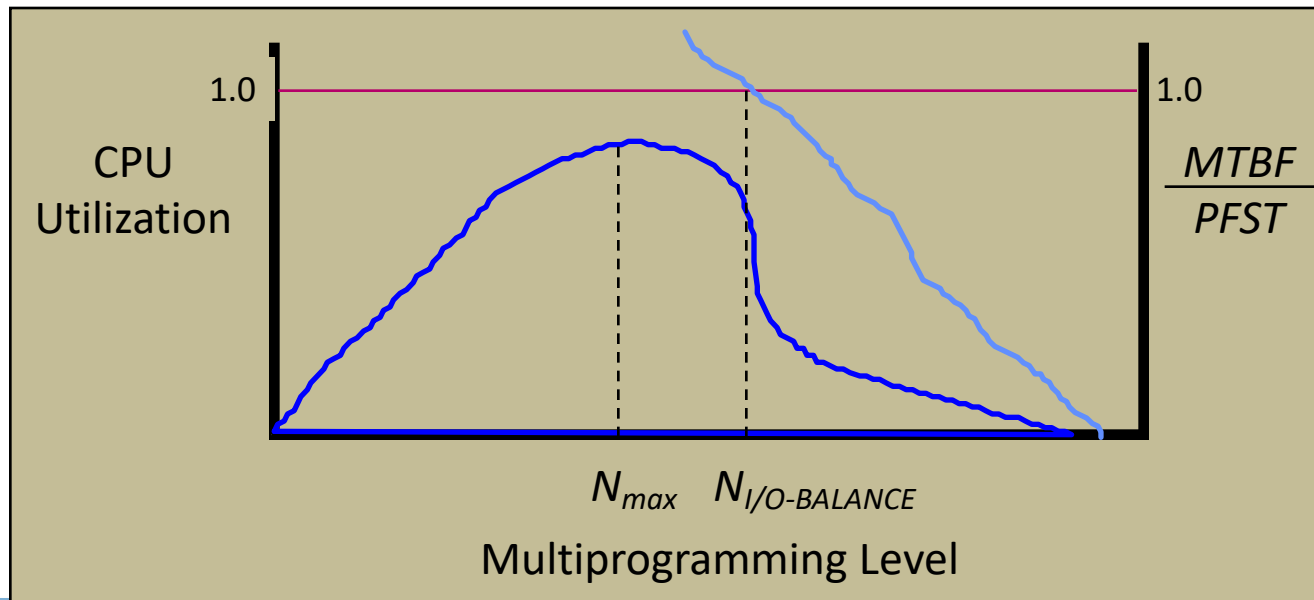


System is *thrashing* — spending all of its time paging



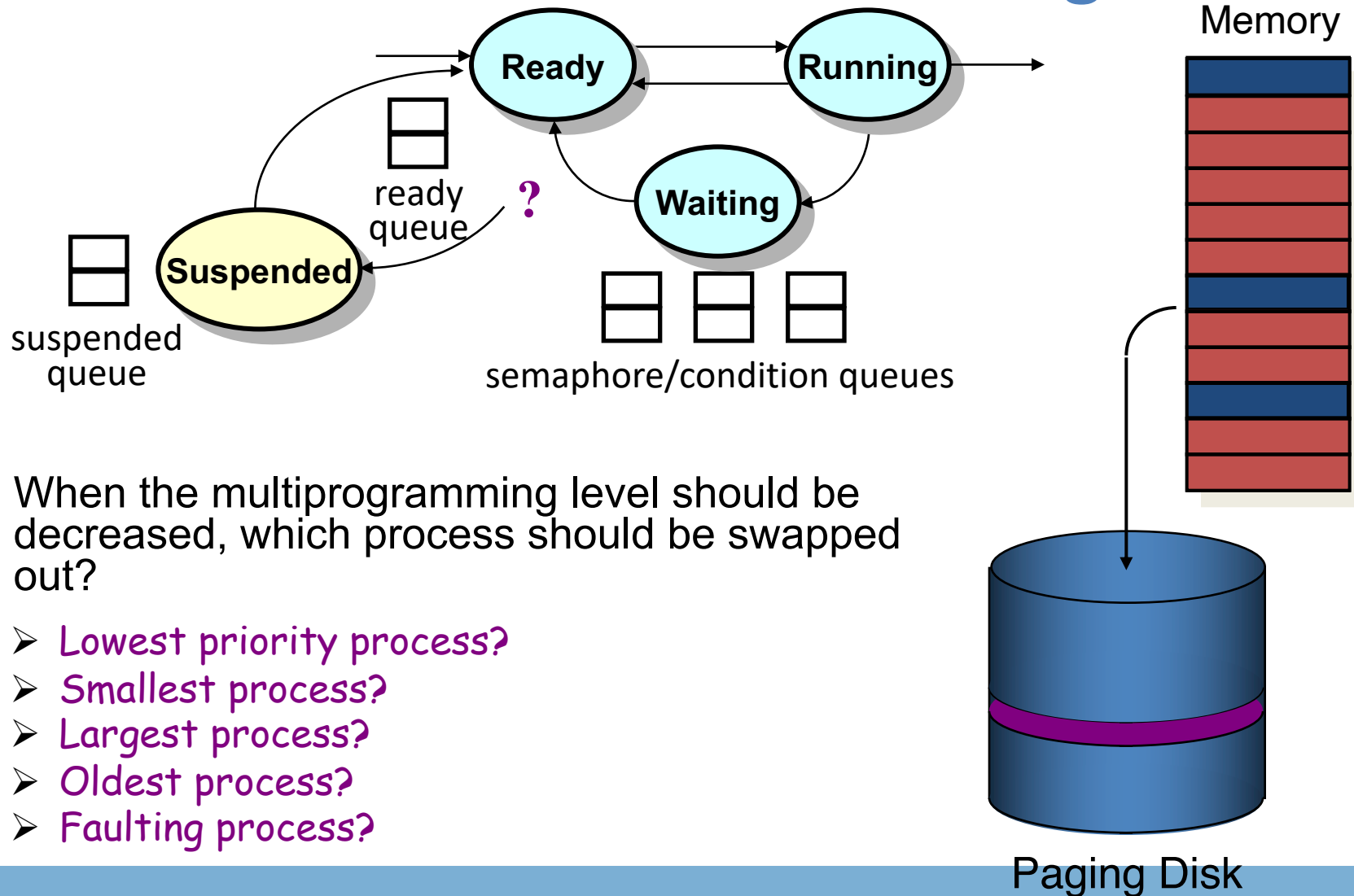
Load Control and Thrashing

- Thrashing can be ameliorated by *local* page replacement
- ◆ Better criteria for load control: Adjust MPL so that:
 - *mean time between page faults (MTBF) = page fault service time (PFST)*
 - $\sum WS_i = \text{size of memory}$





Load Control and Thrashing



- When the multiprogramming level should be decreased, which process should be swapped out?

- **Lowest priority process?**
- **Smallest process?**
- **Largest process?**
- **Oldest process?**
- **Faulting process?**