## Chapter 10: Virtual Memory





## Chapter 10: Virtual Memory

- **Background**
- **Demand Paging**
- **Copy-on-Write**
- **Page Replacement**
- **Allocation of Frames**
- **Thrashing**
- **Nemory-Mapped Files**
- **Allocating Kernel Memory**
- **Diter Considerations**
- **Operating-System Examples**





## **Objectives**

- **Define virtual memory and describe its benefits.**
- **Illustrate how pages are loaded into memory using demand paging.**
- **Apply the FIFO, optimal, and LRU page-replacement algorithms.**
- **Describe the working set of a process, and explain how it is related to** program locality.
- Describe how Linux, Windows 10, and Solaris manage virtual memory.
- Design a virtual memory manager simulation in the C programming<br>
language.<br>
Operating System Concepts 10<sup>th</sup> Edition 10.3 Silberschatz, Galvin and Gagne ©2018 Design a virtual memory manager simulation in the C programming language.





## **Background**

- Code needs to be in memory to execute, but entire program rarely used
	- Error code, unusual routines, large data structures
- Background<br>
 Code needs to be in memory to execute, but entire program rarely used<br>
 Error code, unusual routines, large data structures<br>
 Even in those cases where the entire program is needed, it may not all be<br>
 Con **Example 19 Starting Code Starting Code and Starting Code seds to be in memory to execute, but entire program race the same time**<br>
Firor code, unusual routines, large data structures<br>
Even in those cases where the entire p
- Consider ability to execute partially-loaded program
	- Program no longer constrained by limits of physical memory
	-
- Program and programs could be larger than physical memory<br>
 Each program takes less memory while running -> more programs run<br>
at the same time<br>
 Increased CPU utilization and throughput with no increase in<br>
response model are discomed and the larger than physical memory of the same time of the same time edded at the same time consider ability to execute partially-loaded program (and program of longitum of larger than physical memory a • Each program takes less memory while running -> more programs run at the same time
	- Increased CPU utilization and throughput with no increase in response time or turnaround time





- Virtual memory<br>
Virtual memory separation of user logical memory (virtual address<br>
space) from physical memory<br>
 Only part of the program needs to be in memory for execution Virtual memory<br>Virtual memory – separation of user logical memory<br>space) from physical memory<br>• Only part of the program needs to be in memory<br>• Logical address space can therefore be much Virtual memory<br>
Virtual memory<br>
Space) from physical memory<br>
• Only part of the program needs to be in memory for execution<br>
• Logical address space can therefore be much larger than physical<br>
• Virtual address space<br>
• Vi
	- Only part of the program needs to be in memory for execution
	- Logical address space can therefore be much larger than physical address space
- - Usually start at address 0, contiguous addresses until end of space
- Usually start at address U, contiguous addresses until end of space<br>
 Meanwhile, physical memory organized in page frames<br>
 MMU must map logical to physical<br>
 Virtual memory can be implemented via:<br>
 Demand paging<br> • Meanwhile, physical memory organized in page frames
	- MMU must map logical to physical
	- Virtual memory can be implemented via:
		- Demand paging
		- Demand segmentation



Virtual Memory That is Larger Than Physical Memory





- **Demand Paging<br>
Pages are only loaded when they are demanded during<br>
Pages that are never accessed are thus never loaded into physical<br>
Reges that are never accessed are thus never loaded into physical Demand Paging<br>
Pages are only loaded when they are de<br>
program execution<br>
Pages that are never accessed are thus never<br>
memory Demand Paging<br>Pages are only loaded when they are demanded during<br>program execution<br>a Pages that are never accessed are thus never loaded into physical<br>memory<br>A demand-paging system is similar to a paging system with** 
	- memory
- **Demand Paging**<br> **Example 3** Pages are only loaded when they are demanded during<br> **Example 3** Pages that are never accessed are thus never loaded into physical<br> **Example 5** A demand-paging system is similar to a paging sys swapping Pages are only loaded when they are demanded during<br>program execution<br>a Pages that are never accessed are thus never loaded into physical<br>memory<br>A demand-paging system is similar to a paging system with<br>swapping<br>a Rather nges are only loaded when they are demand<br>param execution<br>Pages that are never accessed are thus never lo<br>memory<br>demand-paging system is similar to a pa<br>rapping<br>Rather than swapping the entire process into me<br>use a lazy sw Depress that are never accessed are thus new<br>memory<br>demand-paging system is similar to a<br>rapping<br>Rather than swapping the entire process into<br>use a lazy swapper<br>Lazy swapper: never swaps a page into mer<br>will be needed<br>Swap
- **□ Rather than swapping the entire process into memory, however, we**<br>
use a lazy swapper. never swaps a page into memory unless that page<br>
will be needed<br>
 Swapper that deals with pages is a **pager**<br>
<br>
Operating System C
	- **Lazy swapper: never swaps a page into memory unless that page** 
		- $\blacksquare$  Swapper that deals with pages is a **pager**





## Demand Paging

**Similar to paging system with** swapping (diagram on right)







- With swapping, pager guesses which pages will be used before swapping out again **Basic Concepts**<br>
If with swapping, pager guesses which pages will be used before swapping<br>
Instead, pager brings in only those pages into memory<br>
If page here already memory resident<br>
If pages needed are already memory re
- Instead, pager brings in only those pages into memory
- How to determine that set of pages?
	- Need new MMU functionality to implement demand paging
- - No difference from non demand-paging
- If page needed and not memory resident
- No difference from non demand-paging<br>
 If page needed and not memory resident<br>
 Need to detect and load the page into memory from storage<br>
 Without changing program behavior<br>
 Without programmer needing to change co Need to detect and load the page into memory from storage
	- ▶ Without changing program behavior
	- Without programmer needing to change code



## Valid-Invalid Bit Valid–Invalid Bit<br>
■ With each page table entry a valid–invalid bit is associated<br>■ ▼ ⇒ in-memory – memory resident alid-Invalid Bit<br>
With each page table entry a valid–invalid bit is associated<br>  $\blacksquare$  **v**  $\Rightarrow$  in-memory – memory resident<br>  $\blacksquare$  **i**  $\Rightarrow$  not-in-memory

- -
	- $\blacksquare$   $\blacksquare$   $\Rightarrow$  not-in-memory
- 
- Valid–Invalid Bit<br>
 With each page table entry a valid–invalid bit is associated<br>
 v⇒ in-memory memory resident<br>
 i⇒ not-in-memory<br>
 Initially valid–invalid bit is set to i on all entries<br>
 During address translati **□** With each page table entry a valid–invalid bit is associated<br> **□ V** ⇒ in-memory – memory resident<br> **□ i** ⇒ not-in-memory<br>
□ Initially valid–invalid bit is set to **i** on all entries<br>
□ During address translation, if v With each page table entry a valid–invalid bit is ass<br>  $\Box \textbf{V} \Rightarrow$  in-memory – **memory resident**<br>  $\Box \textbf{i} \Rightarrow$  not-in-memory<br>
Initially valid–invalid bit is set to **i** on all entries<br>
During address translation, if valid–



## Valid-Invalid Bit

Example of a page table snapshot:



 During MMU address translation, if valid–invalid bit in page table entry is i  $\Rightarrow$  page fault





### Page Table When Some Pages Are Not in Main Memory











## Steps in Handling Page Fault

- 1. If there is a reference to a page, first reference to that page will trap to operating system
	- Page fault
- 2. Operating system looks at another table to decide:
	- Invalid reference  $\Rightarrow$  abort
	- Just not in memory
- 3. Find free frame
- 4. Swap page into frame via scheduled disk operation
- 3. Find free frame<br>
4. Swap page into frame via scheduled disk operation<br>
5. Reset tables to indicate page now in memory<br>
Set validation bit = **v**<br>
6. Restart the instruction that caused the page fault<br>  $\frac{1}{2}$ <br>
Operati 5. Reset tables to indicate page now in memory Set validation bit  $=$  v
	- 6. Restart the instruction that caused the page fault





- 1. Trap to the operating system
- 2. Save the user registers and process state
- 3. Determine that the interrupt was a page fault
- 4. Check that the page reference was legal and determine the location of the page on the disk
- 5. Issue a read from the disk to a free frame:
- a) Wait in a queue for this device until the read request is serviced<br>
b) Wait for the device seek and/or latency time<br>
c) Begin the transfer of the page to a free frame<br>
C) Begin the transfer of the page to a free frame<br> a) Wait in a queue for this device until the read request is serviced
	- b) Wait for the device seek and/or latency time
	- c) Begin the transfer of the page to a free frame



## Stages in Demand Paging (Cont.)

- 6. While waiting, allocate the CPU to some other user
- 7. Receive an interrupt from the disk I/O subsystem (I/O completed)
- 8. Save the registers and process state for the other user
- 9. Determine that the interrupt was from the disk
- 10. Correct the page table and other tables to show page is now in memory
- 11. Wait for the CPU to be allocated to this process again
- 12. Restore the user registers, process state, and new page table, and then<br>
resume the interrupted instruction<br>  $\frac{1}{2}$ <br>
Operating System Concepts 10<sup>th</sup> Edition<br>  $\frac{1}{2}$ <br>  $\frac{1}{2}$ <br>  $\frac{1}{2}$ <br>  $\frac{1}{2}$ <br>  $\frac{1}{2}$ 12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction





## Pure Demand Paging

- Start process with **no** pages in memory
- OS sets instruction pointer to first instruction of process, non-memoryresident  $\Rightarrow$  page fault occurs • **Pure Demand Paging**<br>
• OS sets instruction pointer to first instruction of process, non-memory-<br>
• And for every other process pages, page fault occurs on first access<br>
• Never bring a page into memory until it is requ • Pure Demand Paging<br>
• CS sets instruction pointer to first instruction of process, non-memory-<br>
• CS sets instruction pointer to first instruction of process, non-memory-<br>
• And for every other process pages, page fault
	-
	-
- Theoretically, a given instruction could access multiple pages  $\Rightarrow$  multiple page faults
- Consider fetch and decode of instruction which adds 2 numbers from<br>memory and stores result back to memory<br>
Pain decreased because of locality of reference<br>
 Hardware support needed for demand paging<br>
 Page table with • Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
	- Pain decreased because of **locality of reference**
	- **Hardware support needed for demand paging** 
		- Page table with valid / invalid bit
		- Secondary memory (swap device with **swap space)**
		- Instruction restart



## Performance of Demand Paging Performance of Demand Paging<br>
provide a Page Fault Rate 0  $\leq p \leq 1$ <br>
exercive Access Time (EAT)<br>
EAT =  $(1-p) \times$  memory access +  $p \times$  page fault overhead Formance of Demand Paging<br>
Fault Rate 0 ≤ *p* ≤ 1<br>
ctive Access Time (EAT)<br>
EAT = (1– *p*) × memory access + *p* × page fault overhead<br>
a-fault overhead<br>
Service the nace fault interrust Performance of Demand Pa<br>
<br> **a** Page Fault Rate  $0 \le p \le 1$ <br> **compage-fault overhead**<br> **a** page-fault overhead<br>
1. Service the page-fault interrupt<br>
2. Read in the page (read the page from disk) – lots **Example 1.1**<br> **Example Fault Rate 0**  $\leq p \leq 1$ <br> **Effective Access Time (EAT)**<br> **EAT = (1– p) × memory access + p × page fault overhead**<br> **Page-fault overhead**<br> **Page-fault overhead**<br> **Page-fault interrupt**<br> **Page-fault EXECT COMMON COMMON CONTROVER SURFERENT CONTRENT:**<br>
2. Read in the page fault overhead<br>
2. Read in the page from disk) – lots of time<br>
3. Restart the process<br>
2. Read in the page (read the page from disk) – lots of time<br>

- **Page Fault Rate**  $0 \le p \le 1$
- 

Effective Access Time (EAT)<br>
EAT = (1– *p*) × memory access + *p* × page fault overhead<br>
aage-fault overhead<br>
1. Service the page-fault interrupt<br>
2. Read in the page (read the page from disk) – lots of time<br>
3. Restart EAT =  $(1-p) \times$  memory access +  $p \times$  page fault overhead<br>
2. Service the page-fault interrupt<br>
2. Read in the page (read the page from disk) – lots of time<br>
3. Restart the process<br>
Example<br>
a Memory access time = 200 nano

- -
- Page Fault Rate  $0 \le p \le 1$ <br>
Effective Access Time (EAT)<br>
EAT =  $(1-p) \times$  memory access +  $p \times$  pag<br>
page-fault overhead<br>
1. Service the page-fault interrupt<br>
2. Read in the page (read the page from dis<br>
3. Restart the proc
	-
- **Example** 
	-
	-



## Demand Paging Example **Demand Paging Exam**<br>
Memory access time = 200 nanoseconds<br>
Average page-fault service time = 8 milliseconds<br>
EAT =  $(1 - p) \times 200 + p$  (8 milliseconds)<br>
=  $(1 - p \times 200 + p \times 8,000,000$ <br>
=  $200 + p \times 7,999,800$ **Demand Paging Exam**<br>ory access time = 200 nanoseconds<br>age page-fault service time = 8 milliseconds<br>=  $(1 - p) \times 200 + p$  (8 milliseconds)<br>=  $(1 - p \times 200 + p \times 8,000,000$ <br>=  $200 + p \times 7,999,800$ <br>e access out of 1,000 causes a page

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- 

 $= 200 + p \times 7,999,800$ 

If one access out of 1,000 causes a page fault, then

 $P = 1/1000$ 

 $FAT = 8.2$  microseconds.

This is a slowdown by a factor of 40!! (the access time gets 1/40!!)

- $P = 1/1000$ <br>  $EAT = 8.2$  microseconds.<br>
This is a slowdown by a factor of 40!! (the access time gets 1/40!!)<br>
If want performance degradation < 10 percent, then  $EAT = 200 + 20$  220<br>  $\cdot$  220 > 200 + 7,999,800 x p<br>  $20 > 7,999,$ If want performance degradation  $<$  10 percent, then EAT = 200 + 20 220
	- 220 > 200 + 7,999,800 x p 20 > 7,999,800 x p
	- $p < .0000025$  => (25 page faults in any 10,000,000 memory accesses)
	- < one page fault in every 400,000 memory accesses





## Copy-on-Write

- Recall that the fork() system call creates a child process as a duplicate of its parent
	- It creates a copy of the parent's address space for the child
- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory
	- If either process modifies a shared page, only then is the page copied
- If either process modifies a shared page, only then is the page<br>
copied<br>
 COW allows more efficient process creation as only modified pages are<br>
copied<br>
<br>
Operating System Concepts 10<sup>th</sup> Edition<br>
10.20<br>
Silberschatz COW allows more efficient process creation as only modified pages are copied



# Page Replacement

- Page Replacement<br>
when virtual memory management over-allocates memory, it is possible<br>
that all available memory is used by active processes Page Replacement<br>
When virtual memory management over-allocates memory, it is possible<br>
that all available memory is used by active processes<br>
■ In this situation, if a page fault occurs, there is no free frame to allocat Page Replacement<br>
When virtual memory management over-allocates memory, it is possible<br>
that all available memory is used by active processes<br>
In this situation, if a page fault occurs, there is no free frame to allocate i age Replacement<br>
Nen virtual memory management over-allocates me<br>
t all available memory is used by active processes<br>
In this situation, if a page fault occurs, there is no free f<br>
the requested page<br>
Iution Find some properties are the used by active processes<br>Find all available memory is used by active processes<br>Find some page fault occurs, there is no free frame to allocate it to<br>the requested page<br>Solution<br>Find some page i
	- Their virtual memory management over-allocate<br>the all available memory is used by active proce<br>th this situation, if a page fault occurs, there is no<br>the requested page<br>lution<br>Find some page in memory that is not currently Solution if a page fault occurs, there is no free frame to allocate it to<br>the requested page<br>Solution<br>**Example 18 and page in memory** that is not currently being used and page it out<br>**Example 20** and frame by writing its
- □ Solution
	-
	- We can free a frame by writing its contents to swap space and changing
	-



## Need For Page Replacement





- 
- **Page Replacement<br>
Page replacement increases the effective access time**<br> **D** Use **modify (dirty) bit** to reduce overhead of page transfers<br> **Exact page has a modify bit associated with it** □ Use modify (dirty) bit to reduce overhead of page transfers
	-
	- **Page Replacement**<br>
	Page replacement increases the effective access time<br>
	Jse modify (dirty) bit to reduce overhead of page transfers<br> **a** Each page has a modify bit associated with it<br> **a** The modify bit for a page is set  $\blacksquare$  The modify bit for a page is set by the hardware whenever any word **Page Replacement**<br>ge replacement increases the effective access time<br>e modify (dirty) bit to reduce overhead of page transfers<br>Each page has a modify bit associated with it<br>The modify bit for a page is set by the hardware
	-
- <p>□ Page replacement completes separation between logical memory and physical memory</p>\n<p>□ large virtual memory can be provided on a smaller physical memory</p>\n<p>□ large virtual memory can be provided on a smaller physical memory</p>\n<p>□ Large virtual memory can be provided on a smaller physical memory</p>\n<p>□ Since the number of vertices are not provided by the image.</p>\n<p>□ Since the number of vertices are not provided by the image.</p>\n<p>□ The number of vertices are not provided by the image.</p>\n<p>□ The number of vertices are not provided by the image.</p>\n<p>□ The number of vertices are not provided by the image.</p>\n<p>□ The number of vertices are not provided by the image.</p>\n<p>□ The number of vertices are not provided by the image.</p>\n<p>□ The number of vertices are not provided by the image.</p>\n<p>□ The number of vertices are not provided **Page Replacement**<br>
Page replacement increases the effective access time<br>
Jse **modify (dirty) bit** to reduce overhead of page transfers<br> **Exact page has a modify bit associated with it**<br> **n** The modify bit for a page is se **Page Replacement**<br> **Example 19 and Separation Between Separation Between Separation Between Separation Between Separation between logical memory and physical memory and page replacement completes separation between logica** Page replacement increases the effective acce<br>Use **modify (dirty) bit** to reduce overhead of  $\vert$ <br> **n** Each page has a modify bit associated with it<br> **n** The modify bit for a page is set by the hardware<br>
or byte in the pa Page replacement increases the effective access time<br>
Jse **modify (dirty) bit** to reduce overhead of page transfers<br> **n** Each page has a modify bit associated with it<br> **n** The modify bit for a page is set by the hardware
	-



## **Demand Paging Algorithms** Demand Paging Algorithms<br>
Frame-allocation algorithm<br>
Frame-allocation algorithm<br>
For Determines how many frames to allocate to each process **Demand Paging Algorithms<br>Prame-allocation algorithm<br>a Determines how many frames to allocate to each process<br>Page-replacement algorithm** Demand Paging Algorithms<br>
Frame-allocation algorithm<br> **Example:** Determines how many frames to allocate to each process<br> **Example:** Page-replacement algorithm<br> **Example:** Selects the frames that are to be replaced

- -
- -
- **Semand Paging Algorithms**<br>
Frame-allocation algorithm<br> **a** Determines how many frames to allocate to each process<br>
Page-replacement algorithm<br> **a** Selects the frames that are to be replaced<br>
Designing efficient algorithms **Designing efficient algorithm**<br> **Designing efficient algorithm**<br> **Designing efficient algorithm**<br> **Designing efficient algorithms** is so important, because<br>
disk I/O is so expensive Frame-allocation algorithm<br> **n** Determines how many frames to allocate to each p<br>
Page-replacement algorithm<br> **n** Selects the frames that are to be replaced<br>
Designing efficient algorithms is so important<br>
disk I/O is so e **□** Determines how many frames to allocate to each process<br>Page-replacement algorithm<br>□ Selects the frames that are to be replaced<br>Designing efficient algorithms is so important, because<br>disk I/O is so expensive<br>□ In gen
	-

# Page Replacement Algorithms

- $\Box$  We evaluate algorithm by running it on a particular string of memory Page Replacement Algorithms<br>We evaluate algorithm by running it on a particular string of memory<br>references (reference string) and computing the number of page<br>faults on that string Page Replacement Algorithm<br>We evaluate algorithm by running it on a particular<br>references (reference string) and computing the nu<br>faults on that string<br>■ String is just page numbers, not full addresses Replacement Algorithms<br>We evaluate algorithm by running it on a particular string of memory<br>references (reference string) and computing the number of page<br>faults on that string<br>a String is just page numbers, not full addre Page Replacement Algorithms<br>
We evaluate algorithm by running it on a particular string of memory<br>
references (reference string) and computing the number of page<br>
raults on that string<br>
■ String is just page numbers, not ■ We evaluate algorithm by running it on a particular string of memory<br>references (reference string) and computing the number of page<br>faults on that string<br>■ String is just page numbers, not full addresses<br>■ Repeated acce algorithm by running it on a particular string of memory<br>sference string) and computing the number of page<br>string<br>t page numbers, not full addresses<br>ccess to the same page does not cause a page fault<br>mples, the reference s references (reference string) and computing the num<br>faults on that string<br> $\Box$  String is just page numbers, not full addresses<br> $\Box$  Repeated access to the same page does not cause a p<br>in all our examples, the reference st
	-
	-
- 

## Page Replacement Algorithms Page Replacement Algorithms<br>□ Optimal<br>□ FIFO<br>□ LRU (Least Recently Used)<br>□ LRU Approximation Replacement Algorithms<br>
Additional<br>-RU (Least Recently Used)<br>-RU Approximation<br>- Additional-Reference-Bits Algorithm<br>- Second-Chance Algorithm

- **D** Optimal
- **D**FIFO
- 
- $\Box$  LRU Approximation
	-
	-
- Optimal<br>FIFO<br>LRU (Least Recently Used)<br>LRU Approximation<br>■ Additional-Reference-Bits Algorithm<br>■ Second-Chance Algorithm<br>Counting-Based Page Replacement ם Optimal<br>
DENU (Least Recently Used)<br>
DENU Approximation<br> **Counting-Based Page Replacement<br>
DENU (Least Frequently Used)**<br>
DENU (Least Frequently Used) FIFO<br>LRU (Least Recently Used)<br>LRU Approximation<br>n Additional-Reference-Bits Algorithm<br>n Second-Chance Algorithm<br>Counting-Based Page Replacement<br>n LFU (Least Frequently Used)<br>n MFU (Most Frequently Used) NUMENT (Least Recently Used)<br>
NUMENT Approximation<br>
Additional-Reference-Bits Algorithm<br>
α Second-Chance Algorithm<br>
Counting-Based Page Replacement<br>
α LFU (Least Frequently Used)<br>
α MFU (Most Frequently Used)
	-
	-

## Optimal Page Replacement Optimal Page Replacement<br>
Peplace page that will not be used for longest period of time<br>
Peplace page faults<br>
Replace 9 page faults<br>
Replace you know this? Example: 9 page faults Optimal Page Replacement<br>
Proplace page that will not be used for longest period of<br>
Proplace page faults<br>
Proplace 19 page faults<br>
Proplace the future<br>
Proper Can't read the future<br>
Proper Used for measuring how well your Otimal Page Replacement<br>Replace page that will not be used for longest period<br>Example: 9 page faults<br>How do you know this?<br>a Can't read the future<br>Used for measuring how well your algorithm perform: Optimal Page Replacement<br>
a Replace page that will not be used for longest period of time<br>
a Example: 9 page faults<br>
a How do you know this?<br>
a Can't read the future<br>
a Used for measuring how well your algorithm performs

- 
- 
- -
- 



# First-In-First-Out (FIFO) Algorithm

- First-In-First-Out (FIFO) Algorithm<br>
Moreon a page must be replaced, the oldest page is chosen<br>
Moreon a Can be implemented using a FIFO queue<br>
Reference faults irst-In-First-Out (FIFO) Algorithm<br>When a page must be replaced, the oldest page is chosen<br>a Can be implemented using a FIFO queue<br>Example: 15 page faults First-In-First-Out (FIFO) Algorithm<br>
<del>Example: 15 page faults</del><br>
Example: 15 page faults
- 



Graph of Page Faults Versus the Number of Frames



# Belady's Anomaly

- Belady's Anomaly<br>
For some page-replacement algorithms, the page-fault rate may<br>
increase as the number of allocated frames increases<br>
Example: reference string = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 Belady's Anomaly<br>For some page-replacement algorithms, the page-fault rate may<br>increase as the number of allocated frames increases<br>Example: reference string = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 



# Least Recently Used (LRU)

- Least Recently Used (LRU)<br>  $\Box$  LRU is an approximation of the optimal algorithm<br>  $\Box$  It uses the recent past as an approximation of the near future<br>  $\Box$  LRU replaces the page that has not been used for the **Past Recently Used (LRU)**<br>LRU is an approximation of the optimal algorithm<br>a It uses the recent past as an approximation of the near future<br>LRU replaces the page that has not been used for the<br>ongest period of time
- Least Recently Used (LRU)<br> **Example 18 All Allen Conducts** an approximation of the optimal algorithm<br> **Example 18 Allen Conducts** as an approximation of the near future<br> **EXU** replaces the page that has not been used for t **East Recently Used (LRU)**<br>LRU is an approximation of the optimal algorithm<br>**alt uses the recent past as an approximation of the near i**<br>LRU replaces the page that has not been used filongest period of time



## Least Recently Used (LRU) Least Recently Used (L<br>
Stack implementation<br>
Fine a stack of page numbers in a double line Least Recently Used (LRU)<br>Stack implementation<br>
Exterp a stack of page numbers in a double link form<br>
Exterp a stack of page numbers in a double link form<br>
Retract to the top Least Recently Used (LRU)<br>
Stack implementation<br>
Figure a stack of page numbers in a double link form<br>
Page referenced: move it to the top<br>
But each update more expensive<br>
Nu casual france expensive Least Recently Used (LRU)<br>Stack implementation<br>a Keep a stack of page numbers in a double link form<br>a Page referenced: move it to the top<br>a But each update more expensive<br>a No search for replacement Least Recently Used (LRU)<br>Stack implementation<br>
Exeep a stack of page numbers in a double link form<br>
Exerch for replacement<br>
No search for replacement<br>
Counter implementation

- 
- 
- 
- 
- 
- ECASTINGCCTITY USCU (LI<br>
a Stack implementation<br>
a Keep a stack of page numbers in a double link f<br>
a Page referenced: move it to the top<br>
a But each update more expensive<br>
a No search for replacement<br>
Counter implementati Every a stack of page numbers in a double link form<br> **Every a** stack of page numbers in a double link form<br> **Every counter** implementation<br> **Every page entry has a counter; every time page is referenced through this entry,** Examplementation<br>
Keep a stack of page numbers in a double link form<br>
Page referenced: move it to the top<br>
But each update more expensive<br>
No search for replacement<br>
unter implementation<br>
Every page entry has a counter; ev n Reep a stack of page numbers in a double link form<br>
n Page referenced: move it to the top<br>
n But each update more expensive<br>
n No search for replacement<br>
Counter implementation<br>
n Every page entry has a counter; every t Page referenced: move it to the top<br>But each update more expensive<br>No search for replacement<br>Linter implementation<br>Every page entry has a counter; every time page is referenced thr<br>copy the clock into the counter<br>When a pa
	- -

# Least Recently Used (LRU)

- Least Recently Used (LRU)<br>— LRU and OPT are cases of stack algorithms that don't<br>— have Belady's Anomaly have Belady'sAnomaly
- An algorithm for which it can be shown that the set of pages in ast Recently Used (LRU)<br>RU and OPT are cases of **stack algorithms** that don't<br>ve Belady's Anomaly<br>An algorithm for which it can be shown that the set of pages in<br>memory for n frames is always a subset of the set of pages t **SIME RECENTITY USEG (LRU)**<br>Wand OPT are cases of **stack algorithms** that don't<br>ve Belady's Anomaly<br>An algorithm for which it can be shown that the set of pages in<br>memory for n frames is always a subset of the set of pages <p>□ LRU and OPT are cases of <b>stack algorithms</b> that don't have <b>Belady's</b> Anomaly</p>\n<p>□ An algorithm for which it can be shown that the set of pages in memory for n frames is always a subset of the set of pages that would be in memory with n + 1 frames</p>\n<p>□ Implementation of LRU needs hardware assistance</p>\n<p>□ The updating of the clock fields or stack must be done for every</p> LRU and OPT are cases of **stack algorithms** that don't<br>have Belady's Anomaly<br>a An algorithm for which it can be shown that the set of pages in<br>memory for n frames is always a subset of the set of pages that<br>would be in mem ve Belady's Anomaly<br>An algorithm for which it can be shown that the<br>memory for n frames is always a subset of the<br>would be in memory with n + 1 frames<br>plementation of LRU needs hardware<br>The updating of the clock fields or
- -

## LRU ApproximationAlgorithms LRU Approximation Algorithms<br>
FILRU needs special hardware and still slow<br>
Few computer systems provide sufficient hardware support for true LRU FRU Approximation Algorithms<br>FRU needs special hardware and still slow<br>Few computer systems provide sufficient hardware support for true LRU<br>FRU Many systems provide some help, in the form of a reference bit

- 
- 
- 
- LRU Approximation Algorithms<br>
FRU needs special hardware and still slow<br>
Few computer systems provide sufficient hardware support for true LRU<br>
FRU Many systems provide some help, in the form of a reference bit<br>
FRU The re  $\blacksquare$  The reference bit for a page is set by the hardware whenever that page is referenced RU needs special hardware and still slow<br>Few computer systems provide sufficient hardware support for true LRU<br>Many systems provide some help, in the form of a reference bit<br>a The reference bit for a page is set by the har
	-

## Additional-Reference-Bits Algorithm

- 
- Additional-Reference-Bits Algorithm<br>
Fracep an 8-bit byte for each page<br>
At regular intervals (say, 100 ms), a timer interrupt transfers contrate operating system Additional-Reference-Bits Algorithm<br>
Fracep an 8-bit byte for each page<br>
At regular intervals (say, 100 ms), a timer interrupt transfers control to<br>
the operating system<br>
Frace person shifts the reference bit for each page Additional-Reference-Bits Alg<br>Keep an 8-bit byte for each page<br>At regular intervals (say, 100 ms), a timer interrupt t<br>the operating system<br>The operating system shifts the reference bit for ea<br>high-order bit of its 8-bit b Additional-Reference-Bits Algorithm<br>
Frame Reep an 8-bit byte for each page<br>
At regular intervals (say, 100 ms), a timer interrupt transfers control to<br>
the operating system<br>
The operating system shifts the reference bit f Additional-Reference-Bits Algorithm<br>Keep an 8-bit byte for each page<br>At regular intervals (say, 100 ms), a timer interrupt transfers control to<br>the operating system<br>The operating system shifts the reference bit for each pa
- 
- Additional-Reference-Bits Algorithm<br>  $\Box$  Keep an 8-bit byte for each page<br>  $\Box$  At regular intervals (say, 100 ms), a timer interrupt transfers control to<br>
the operating system<br>  $\Box$  The operating system shifts the refer Keep an 8-bit byte for each page<br>At regular intervals (say, 100 ms), a timer interrupt tr<br>the operating system<br>The operating system shifts the reference bit for eac<br>high-order bit of its 8-bit byte, shifting the other bit Keep an 8-bit byte for each page<br>At regular intervals (say, 100 ms), a timer interrupt transfers control to<br>the operating system<br>The operating system shifts the reference bit for each page into the<br>nigh-order bit of its 8-**Example 18 and S-bit byte for each page**<br> **Example 18 at regular intervals (say, 100 ms), a timer interrupt transfers control to**<br>
the operating system<br> **Example:** The operating system shifts the reference bit for each

replaced.

## Second Chance Algorithm Second Chance Algorithm<br>■<br>■ The basic algorithm is FIFO<br>■ When a page has been selected, we inspect its refere

- 
- Second Chance Algorithm<br>
The basic algorithm is FIFO<br>
Mhen a page has been selected, we inspect its reference bit<br>
Fifthe value is 0, we proceed to replace this page
	-
- Second Chance Algorithm<br>
The basic algorithm is FIFO<br>
When a page has been selected, we inspect its reference bit<br>
In If the value is 0, we proceed to replace this page<br>
In If the value is 1, we give the page a second chan If the value is 1, we give the page a second chance and move on to select of the next FIFO<br>select the next algorithm is FIFO<br>then a page has been selected, we inspect its reference bit<br>If the value is 0, we proceed to replace this page<br>If the value is 1, we give the page a second chance a ie basic algorithm is FIFO<br>hen a page has been selected, we inspect its re<br>If the value is 0, we proceed to replace this page<br>If the value is 1, we give the page a second chance and<br>select the next FIFO page, its reference The basic algorithm is FIFO<br>  $\Box$  When a page has been selected, we ins<br>  $\Box$  If the value is 0, we proceed to replace this p<br>  $\Box$  If the value is 1, we give the page a second of<br>
select the next FIFO page, its reference When a page has been selected, we inspect its reference bit<br>
If the value is 0, we proceed to replace this page<br>
If the value is 1, we give the page a second chance and move on to<br>
select the next FIFO page, its reference If the value is 0, we proceed to replace this parties in the value is 1, we give the page a second conselect the next FIFO page, its reference bit is time is reset to the current<br>ock algorithm<br>An implementation of the seco
- -

## Second Chance Algorithm



# Second Chance Algorithm



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

### Figure 8.16 Example of Clock Policy Operation

## Enhanced Second-Chance Algorithm<br>example an use reference bit and modify bit together Enhanced Second-Chance Algorithm<br>
Me can use reference bit and modify bit together<br>
Then we have four cases<br>
(0, 0) neither recently used nor modified - best page to replace Enhanced Second-Chance Algo<br>  $\Box$  We can use reference bit and modify bit together<br>  $\Box$  Then we have four cases<br>  $\Box$  (0, 0) neither recently used nor modified - best page to repla<br>  $\Box$  (0, 1) not recently used but modi

- 
- -
- (1) netable of Second-Chance Algorithm<br>We can use reference bit and modify bit together<br>Then we have four cases<br>(1) (0, 0) neither recently used nor modified best page to replace<br>(1) (0, 1) not recently used but modified (1, 0) recently used but clear - probably will be used again soon<br>  $\mu$  can use reference bit and modify bit together<br>
Then we have four cases<br>  $\mu$  (0, 0) neither recently used nor modified - best page to replace<br>  $\mu$  (0 **hanced Second-Chance**<br> **hanced Second-Chance**<br> **e** can use reference bit and modify bit togeth<br>
en we have four cases<br>
(0, 0) neither recently used nor modified - best pa<br>
(0, 1) not recently used but modified - not quite Manced Second-Chance Algorithm<br>We can use reference bit and modify bit together<br>Then we have four cases<br>a (0, 0) neither recently used nor modified- best page to replace<br>a (0, 1) not recently used but modified- not quite a e can use reference bit and modify bit together<br>en we have four cases<br>(0, 0) neither recently used nor modified - best page to replace<br>(0, 1) not recently used but modified- not quite as good, need to<br>written out<br>(1, 0) re
	-
- (1,1) recently used but clean probably will be used again soon,<br>and modified probably be used and modified probable probable and (0, 0) neither recently used nor modified best page to replace<br>a (0, 0) net recently
- $\Box$  We replace the first page encountered in the lowest nonempty class

## Enhanced Second-Chance Algorithm



Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]



 $F$  = page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page Replacement Algorithms

## **Comparison**



Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

# Counting Algorithms

Counting Algorithms<br>□ Keep a counter of the number of references that have been<br>made to each page Counting Algorithms<br>Keep a counter of the number of references t<br>made to each page<br>Least Frequently Used (LFU) Counting Algorithms<br>
■ Keep a counter of the number of references that have<br>
made to each page<br>
■ Least Frequently Used (LFU)<br>
■ Requires that the page with the smallest count be replaced Counting Algorithms<br>Keep a counter of the number of references that have been<br>made to each page<br>Least Frequently Used (LFU)<br>a Requires that the page with the smallest count be replaced<br>a Problem: when a page is used heavil COUTITING AIGOTITITITIS<br>
Keep a counter of the number of references that have been<br>
made to each page<br>
Least Frequently Used (LFU)<br>
■ Requires that the page with the smallest count be replaced<br>■ Problem: when a page is us Neep a counter of the number of references that have been<br>made to each page<br>Least Frequently Used (LFU)<br>a Requires that the page with the smallest count be replaced<br>a Problem: when a page is used heavily during the initial

- -
	- process but then is never used again
	-

# Counting Algorithms

- 
- Counting Algorithms<br>□ Most Frequently Used (MFU)<br>□ is based on the argument that the page with the smallest cou Counting Algorithms<br>Most Frequently Used (MFU)<br>is based on the argument that the page with the smallest count was<br>probably just brought in and has yet to be used Produnting Algorithms<br>post Frequently Used (MFU)<br>is based on the argument that the page with the smallest count was<br>probably just brought in and has yet to be used<br>either MFU nor LFU replacement is common Counting Algorithms<br>
■ Most Frequently Used (MFU)<br>
■ is based on the argument that the page with the smallest count was<br>
probably just brought in and has yet to be used<br>
■ Neither MFU nor LFU replacement is common<br>
■ The COUNTING AIGOFITNMS<br>
Most Frequently Used (MFU)<br> **a** is based on the argument that the page with the smallest count was<br>
probably just brought in and has yet to be used<br>
Neither MFU nor LFU replacement is common<br> **a** The i Most Frequently Used (MFU)<br> **a** is based on the argument that the page with the smallest count was<br>
probably just brought in and has yet to be used<br>
Neither MFU nor LFU replacement is common<br> **a** The implementation of thes
- -
	-

## Page-Buffering Algorithms Page-Buffering Algorithms<br>
Frames, always<br>
Frames a pool of free frames, always<br>
Frames is chosen as before<br>
Frames a proper the desired page is read into a free frame from the pool

- 
- **⊃age-Buffering Algorithms**<br>Keep a pool of free frames, always<br>■ When a page fault occurs, a victim frame is chosen as before<br>■ However, the desired page is read into a free frame from the pool<br>before the victim is writte **However, the desired page is read into a free frame from the pool** age-Buffering Algorithms<br>eep a pool of free frames, always<br>When a page fault occurs, a victim frame is chosen as before<br>However, the desired page is read into a free frame from the pod<br>When the victim is later written, its **Page-Buffering Algorithms**<br>
Keep a pool of free frames, always<br> **a** When a page fault occurs, a victim frame is chosen as before<br> **a** However, the desired page is read into a free frame from the pool<br> **a** When the victim ■ Keep a pool of free frames, always<br>
■ When a page fault occurs, a victim frame is chosen as before the victim is written out<br>
■ However, the desired page is read into a free frame from the before the victim is written o When a page fault occurs, a victim frame is chosen as before<br> **Example 1998** Molever, the desired page is read into a free frame from the pool<br> **Example 1998** When the victim is written out<br> **Example 1998** When the victim
	- pool
- - non-dirty

# Resident Set Management

- Resident Set Management<br>
 The OS must decide how many pages to bring into main<br>
 memory memory sident Set Management<br>The Management<br>The smaller the amount of memory allocated to each process,<br>The smaller the amount of memory allocated to each process,<br>the more processes that can reside in memory.<br>Small number of pag Chinal Set Management<br>The OS must decide how many pages to bring into main<br>memory<br>a The smaller the amount of memory allocated to each process,<br>the more processes that can reside in memory.<br>a Small number of pages loaded The OS must decide how many pages to bring into main<br>memory<br> **E** The smaller the amount of memory allocated to each process,<br>
the more processes that can reside in memory.<br> **E** Small number of pages loaded increases page f The Solomont decide how many pages to brinct the smaller the amount of memory allocated to each<br>the more processes that can reside in memory.<br>Small number of pages loaded increases page fault<br>Beyond a certain size, further
	- **n** The smaller the amount of memory allocated to each process,
	-
	-

# Resident Set Size

### □ Fixed-allocation

- 
- esident Set Size<br>Fixed-allocation<br>□ Gives a process a fixed number of pages within which to execute<br>□ When a page fault occurs, one of the pages of that process esident Set Size<br>Fixed-allocation<br>■ Gives a process a fixed number of pages within which to execute<br>■ When a page fault occurs, one of the pages of that process<br>must be replaced must be replaced
- **D** Variable-allocation
- Fixed-allocation<br>
a Gives a process a fixed number of pages within which to execute<br> **a** When a page fault occurs, one of the pages of that process<br>
must be replaced<br>
Variable-allocation<br> **a** Number of pages allocated to a ked-allocation<br>Gives a process a fixed number of pages w<br>When a page fault occurs, one of the pages<br>must be replaced<br>riable-allocation<br>Number of pages allocated to a process var<br>of the process

# Fixed Allocation

- Fixed Allocation<br>
Equal allocation<br>
Fifthere are 100 frames (after allocating fra IXed Allocation<br>Equal allocation<br>If there are 100 frames (after allocating frames for the OS) and 5<br>processes, give each process 20 frames processes, give each process 20 frames Fixed Allocation<br>
Fixed Allocation<br>
Figure 200 frames (after allocating frames for<br>
processes, give each process 20 frames<br>
Proportional allocation<br>
Reflection<br>
Reflection<br>
Reflection<br>
Reflection<br>
Reflection **ixed Allocation**<br> **Equal allocation**<br> **I** If there are 100 frames (after allocating frames for the OS) and 5<br>
processes, give each process 20 frames<br>
Proportional allocation<br> **a** Allocate according to the size of process
- -
- $S^{(1)}$   $a_2 = \frac{1}{1}$  $a_i$  = allocationfor  $p_i = \frac{s_i}{s} \times m$  $\frac{q}{q} \times m$   $a = \frac{127}{9} \times 64 \approx 59$ Equal allocation<br> **s** If there are 100 frames (after allocating frames f<br>
processes, give each process 20 frames<br>
Proportional allocation<br> **a** Allocate according to the size of process<br>  $s_i$  = size of process  $p_i$ <br>  $S = \sum s$  $S = \sum s_i$ **a** if there are 100 frames (after allocating frames for the OS<br>
processes, give each process 20 frames<br>
Proportional allocation<br> **a** Allocate according to the size of process<br>  $s_i$  = size of process  $p_i$ <br>  $s_i$  = size of  $S_i$  is m  $s_1 = 10$  $s_2 = 127$  $1^{\circ}$  137<sup>^</sup>  $\sim$  5<sup>1</sup>  $\sim$  5<sup>1</sup>  $a_1 = \frac{10}{10} \times 64 \approx 5$  $\frac{10}{2}$  × 64  $\approx$  5  $\times$  64  $\approx$  5  $2^2 - 137$   $(9 + 8)$  $a_2 = \frac{127}{127} \times 64 \approx 59$  $\frac{127}{126}$  × 64 × 50  $\times$  64  $\approx$  59  $m = 64$

# Replacement Scope

- Replacement Scope<br>
I<br>
In The scope of a replacement strategy can be categorized<br>
as *global* or *local*. as global or local. The scope of a replacement strategy can be categorized<br>as *global* or *local.*<br><br>**a** Both types are activated by a page fault when there are no free<br>page frames.<br><br>**a** A local replacement policy chooses only among the reside
	- **Both types are activated by a page fault when there are no free** page frames.
	- A local replacement policy chooses only among the resident pages of the process that generated the page fault
	- global or local.<br>Both types are activated by a page fault wher<br>page frames.<br>A local replacement policy chooses only amo<br>of the process that generated the page fault<br>A global replacement policy considers all unlo<br>in main me

# Fixed Allocation, Local Scope

- Fixed Allocation, Local Scope<br>Decide ahead of time the amount of allocation to give a<br>process process If allocation is too small, there will be a high page fault rate
- 
- Fixed Allocation, Local Scope<br>
 Decide ahead of time the amount of allocation to give a<br>
 If allocation is too small, there will be a high page fault rate<br>
 If allocation is too large there will be too few programs in<br> Decide ahead of time the amount of<br>process<br>If allocation is too small, there will be<br>If allocation is too large there will be<br>main memory<br>a Increased processor idle time or Decide ahead of time the amount of allocation to give a<br>process<br>If allocation is too small, there will be a high page fault rate<br>If allocation is too large there will be too few programs in<br>main memory<br>**n** Increased proces process<br>If allocation is too small, there will be a hig<br>If allocation is too large there will be too fe<br>main memory<br>a Increased processor idle time or<br>a Increased swapping
	-
	-

## Variable Allocation, Global Scope Variable Allocation, Global S<br>
Easiest to implement<br>
Easiest to implement<br>
Easiest by many operating systems Variable Allocation, Global Scope<br>Easiest to implement<br>a Adopted by many operating systems<br>Operating system keeps list of free frames Variable Allocation, Global Scope<br>
Easiest to implement<br>
Adopted by many operating systems<br>
Operating system keeps list of free frames<br>
Get Free frame is added to resident set of process when a

- -
- 
- Variable Allocation, Global Scope<br>
Easiest to implement<br>
Adopted by many operating systems<br>
Operating system keeps list of free frames<br>
Free frame is added to resident set of process when a<br>
page fault occurs Easiest to implement<br> **n** Adopted by many operating systems<br>
Operating system keeps list of free frame:<br>
Free frame is added to resident set of pro<br>
page fault occurs<br>
If no free frame, replaces one from anothe <p>□ <b>Easiest to implement</b></p>\n<p>□ <b>Adopted by many operating systems</b></p>\n<p>□ <b>Operating system keeps list of free frames</b></p>\n<p>□ <b>Free frame is added to resident set of process when a page fault occurs</b></p>\n<p>□ <b>If no free frame, replaces one from another process</b></p>\n<p>□ <b>Therein lies the difficulty ... which to replace.</b></p> ■ Adopted by many operating systems<br>Operating system keeps list of free frames<br>Free frame is added to resident set of process when a<br>bage fault occurs<br>If no free frame, replaces one from another process<br>■ Therein lies the
- -

# Variable Allocation, Local Scope

- Variable Allocation, Local Scope<br>□ When new process added, allocate number of page<br>frames based on application type, program request, or ariable Allocation, Local Scope<br>When new process added, allocate number of page<br>frames based on application type, program request, or<br>other criteria ariable Allocation, Local<br>When new process added, allocate r<br>frames based on application type, pr<br>other criteria<br>When page fault occurs, select pag Variable Allocation, Local Scope<br>
may When new process added, allocate number of page<br>
frames based on application type, program request, or<br>
other criteria<br>
may When page fault occurs, select page from among<br>
the resident When new process added, allocate number of page<br>frames based on application type, program request, or<br>other criteria<br>When page fault occurs, select page from among<br>the resident set of the process that suffers the fault<br>Ree <p>□ When new process added, allocate number of page frames based on application type, program request, or other criteria</p>\n<p>□ When page fault occurs, select page from among the resident set of the process that suffers the fault</p>\n<p>□ Recaluate allocation from time to time</p>
- 
- 

# Resident Set Management Summary

### Table 8.5 Resident Set Management



## **Thrashing**

- 
- Thrashing<br>■<br>■ Thrashing = a process is busy swapping pages in and out<br>■ If a process does not have "enough" pages, the page-fault Thrashing<br>□ Thrashing = a process is busy swapping pages in and out<br>□ If a process does not have "enough" pages, the page-fault<br>rate is very high Thrashing<br>Thrashing ≡ a process is busy swapping p<br>If a process does not have "enough" pages<br>rate is very high<br>□ Page fault to get page Thrashing<br>
Thrashing = a process is busy swapping pages<br>
If a process does not have "enough" pages, the<br>
rate is very high<br> **n** Page fault to get page<br> **n** Replace existing frame Thrashing = a process is busy swapping pages<br>If a process does not have "enough" pages, the<br>rate is very high<br>a Page fault to get page<br>**D** Replace existing frame<br>**D** But quickly need replaced frame back **Thrashing** = a process is busy swapping pages in and out<br>
If a process does not have "enough" pages, the page-fault<br>
rate is very high<br> **a** Page fault to get page<br> **a** Replace existing frame<br> **a** But quickly need replace
	-
	-
	-

## **Thrashing**



degree of multiprogramming

## End of Chapter 10

