

## Virtual Memory Management

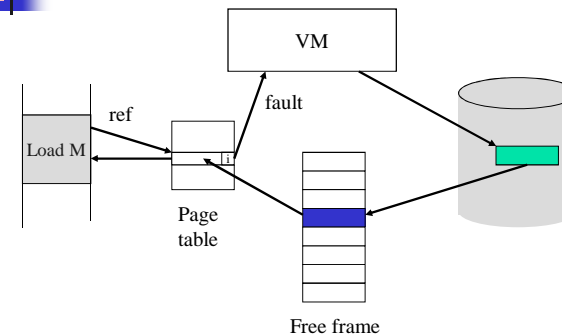
## Fetch: Demand Paging

- Algorithm never brings a page into primary memory until its needed.
  - Page fault
  - Check if a valid virtual memory address. Kill job if not.
  - Find a free page frame.
  - Map address into disk block and fetch disk block into page frame. Suspend user process.
  - When disk read finished, add vm mapping for page frame.
  - Restart instruction.

## Paging Policies

- Fetch Strategies
  - When should a page be brought into primary (main) memory from secondary (disk) storage.
- Placement Strategies
  - When a page is brought into primary storage, where is it to be put?
- Replacement Strategies
  - Which page in primary storage is to be removed when some other page or segment is to be brought in and there is not enough room.

## Demand Paging Example



## Page Replacement

1. Find location of page on disk
2. Find a free page frame
  1. If free page frame use it
  2. Otherwise, select a page frame using the page replacement algorithm
  3. Write the selected page to the disk and update any necessary tables
3. Read the requested page from the disk.
4. Restart instruction.

## Terminology

- **Reference string:** the memory reference sequence generated by a program.
- **Paging** – moving pages to (from) disk
- **Optimal** – the best (theoretical) strategy
- **Eviction** – throwing something out
- **Pollution** – bringing in useless pages/lines

## Issue: Eviction

- Hopefully, kick out a less-useful page
  - Dirty pages require writing, clean pages don't
    - Hardware has a dirty bit for each page frame indicating this page has been updated or not
  - Where do you write? To "swap space" on disk.
- Goal: kick out the page that's least useful
- Problem: how do you determine utility?
  - Heuristic: temporal locality exists
  - Kick out pages that aren't likely to be used again

## Page Replacement Strategies

- **The Principle of Optimality**
  - Replace the page that will not be used the most time in the future.
- **Random page replacement**
  - Choose a page randomly
- **FIFO - First in First Out**
  - Replace the page that has been in primary memory the longest
- **LRU - Least Recently Used**
  - Replace the page that has not been used for the longest time
- **LFU - Least Frequently Used**
  - Replace the page that is used least often
- **Second Chance**
  - An approximation to LRU.

## Principle of Optimality

- Description:
  - Assume that each page can be labeled with the number of instructions that will be executed before that page is first referenced, i.e., we would know the future reference string for a program.
  - Then the optimal page algorithm would choose the page with the highest label to be removed from the memory.
- Impractical because it needs to know future references

## FIFO

12 references,  
9 faults

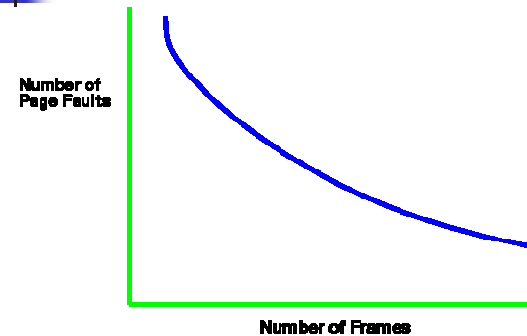
Page Refs	3 Page Frames		
	Fault?	Page Contents	
A	yes	A	
B	yes	B	A
C	yes	C	B A
D	yes	D	C B
A	yes	A	D C
B	yes	B	A D
E	yes	E	B A
A	no	E	B A
B	no	E	B A
C	yes	C	E B
D	yes	D	C E
E	no	D	C E

## Optimal Example

12 references,  
7 faults

Page Refs	3 Page Frames		
	Fault?	Page Contents	
A	yes	A	
B	yes	B	A
C	yes	C	B A
D	yes	D	B A
A	no	D	B A
B	no	D	B A
E	yes	E	B A
A	no	E	B A
B	no	E	B A
C	yes	C	E B
D	yes	D	C E
E	no	D	C E

## Average Paging Behavior with Increasing Page Frames



## Belady's Anomaly (for FIFO)

FIFO with 4  
physical pages

12 references,  
10 faults

As the number of  
page frames  
increase, so does  
the fault rate.

Page Refs	4 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	no	D	C	B
B	no	D	C	B
E	yes	E	D	C
A	yes	A	E	D
B	yes	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

## Least Recently Used Issues

### How to track "recency"?

- use time
  - record time of reference with page table entry
  - use counter as clock
  - search for smallest time.
- use stack
  - remove reference of page from stack (linked list)
  - push it on top of stack
- both approaches require large processing overhead, more space, and hardware support.

## LRU

12 references,  
10 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	A	E	B
B	no	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

## Second Chance

- Only one reference bit in the page table entry.
  - 0 initially
  - 1 When a page is referenced
- pages are kept in FIFO order using a circular list.
- Choose "victim" to evict
  - Select head of FIFO
  - If page has reference bit set, reset bit and select next page in FIFO list.
  - keep processing until you reach page with zero reference bit and page that one out.
- System V uses a variant of second chance

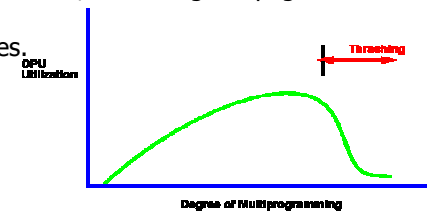
## Second Chance Example

12 references,  
9 faults

Page Refs	3 Page Frames Fault?	Page Contents		
		A*	B*	C*
A	yes	A*		
B	yes	B*	A*	
C	yes	C*	B*	A*
D	yes	D*	C	B
A	yes	A*	D*	C
B	yes	B*	A*	D*
E	yes	E*	B	A
A	no	E*	B	A*
B	no	E*	B*	A*
C	yes	C*	E	B
D	yes	D*	C*	E
E	no	D*	C*	E*

## Thrashing and CPU Utilization

- As the page rate goes up, processes get suspended on page out queues for the disk.
- the system may try to optimize performance by starting new jobs.
- starting new jobs will reduce the number of page frames available to each process, increasing the page fault requests.
- system throughput plunges.

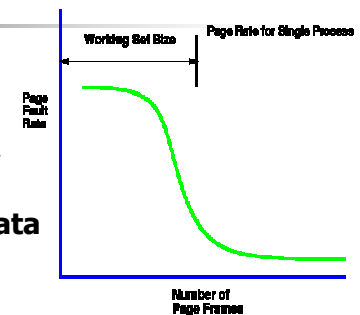


## Thrashing

- Computations have locality.
- As page frames decrease, the page frames available are not large enough to contain the locality of the process.
- The processes start faulting heavily.
- Pages that are read in, are used and immediately paged out.

## Working Set

- the working set model assumes locality.
- the principle of locality states that a program clusters its access to data and text temporally.**
- As the number of page frames increases above some threshold, the page fault rate will drop dramatically.





## Page Size Considerations

- Small pages
  - Reason:
    - Locality of reference tends to be small (256)
    - Less fragmentation
  - Problem: require large page tables
- Large pages
  - Reason
    - Small page table
    - I/O transfers have high seek time, so better to transfer more data per seek
  - Problem: Internal fragmentation, needless caching