COT 4600 Operating Systems Fall 2009

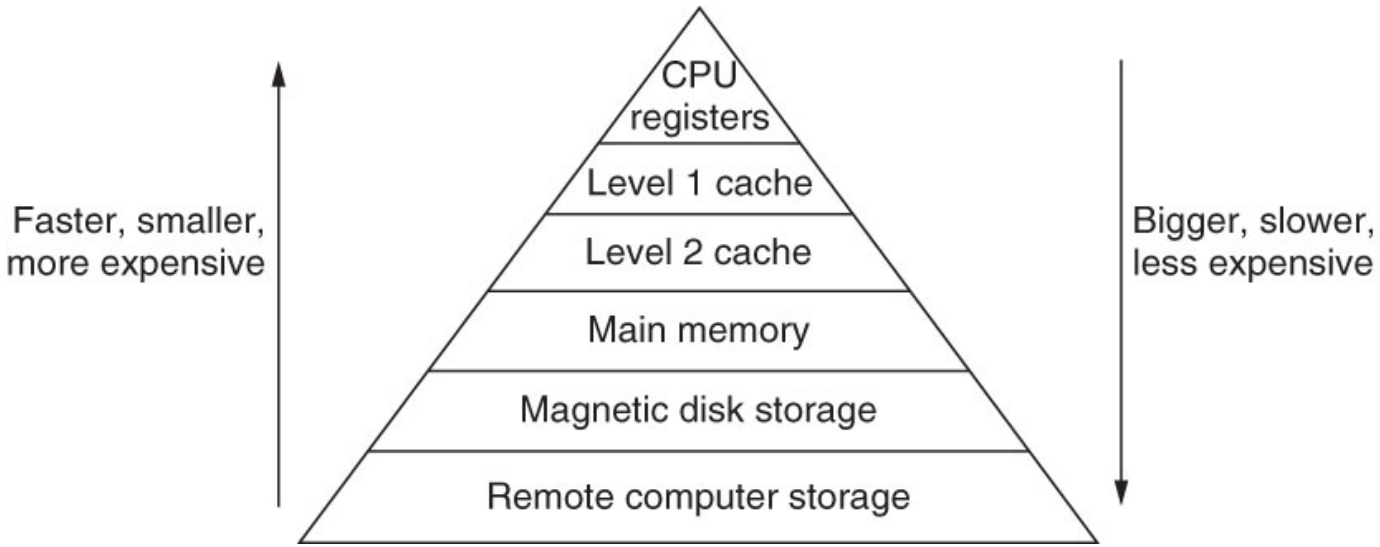
Dan C. Marinescu

Office: HEC 439 B

Office hours: Tu-Th 3:00-4:00 PM

Lecture 24

- Attention: project phase 4 – due Tuesday November 24
  - Final exam – Thursday December 10 4-6:50 PM
- Last time:
  - Methods to diminish the effect of bottlenecks: batching, dallying, speculation
  - I/O systems; the I/O bottleneck
- Today:
  - Multi-level memories
  - Memory characterization
  - Multilevel memories management using virtual memory
  - Adding multi-level memory management to virtual memory
- Next Time:
  - Scheduling


CPU registers
Level 1 cache
Level 2 cache
Main memory
Magnetic disk storage
Remote computer storage
Faster, smaller, more expensive
Bigger, slower, less expensive

Multi-level memories

- In the following hierarchy the amount of storage and the access time increase at the same time
  - CPU registers
  - L1 cache
  - L2 cache
  - Main memory
  - Magnetic disk
  - Mass storage systems
  - Remote storage
- Memory management schemes → where the data is placed through this hierarchy
  - Manual → left to the user
  - Automatic → based on memory virtualization
    - More effective
    - Easier to use

Forms of memory virtualization

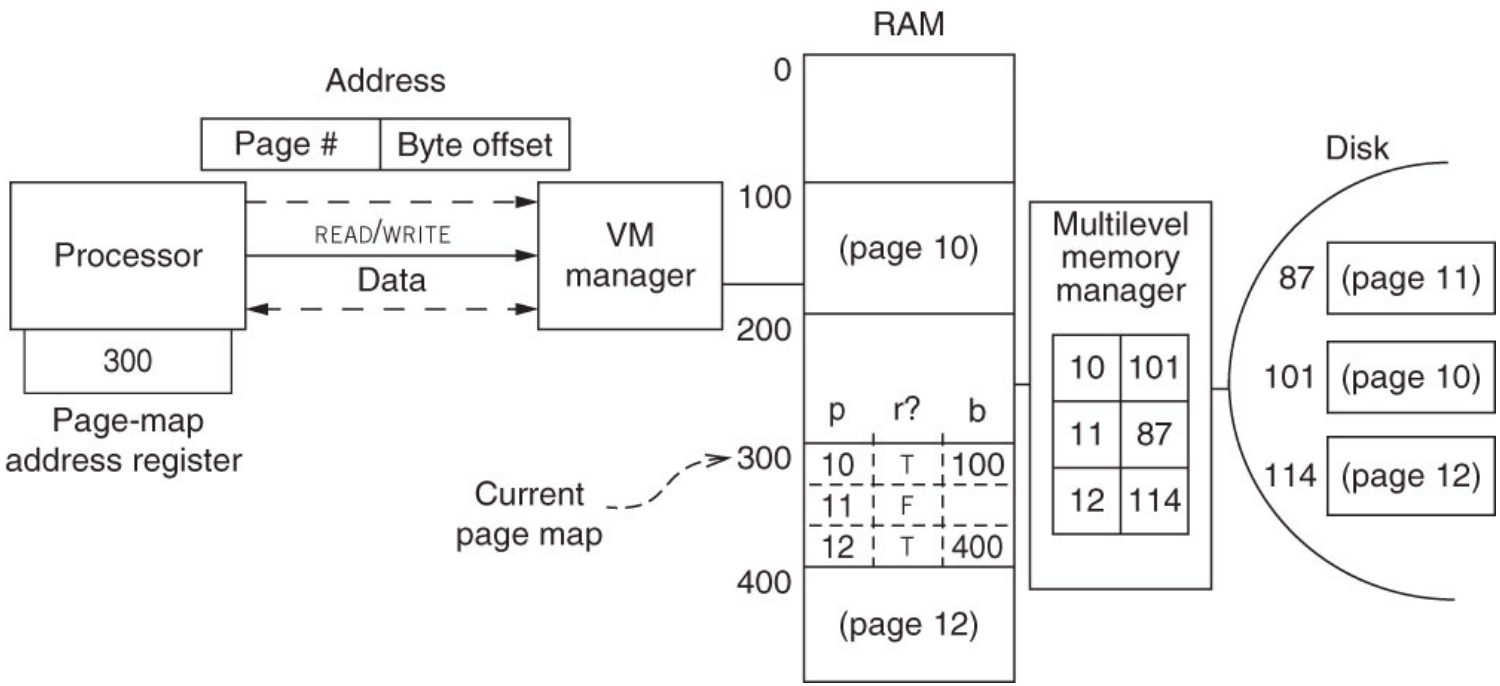
- Memory-mapped files → in UNIX *mmap*
- Copy on write → when several threads use the same data map the page holding the data and store the data only once in memory. This works as long all the threads only READ the data. If one of the threads carries out a WRITE then the virtual memory handling should generate an exception and data pages to be remapped so that each thread gets its only copy of the page.
- On-demand zero filled pages→ Instead of allocating zero-filled pages on RAM or on the disk the VM manager maps these pages without READ or WRITE permissions. When a thread attempts to actually READ or WRITE to such pages then an exception is generated and the VM manager allocates the page dynamically.
- Virtual-shared memory → Several threads on multiple systems share the same address space. When a thread references a page that is not in its local memory the local VM manager fetches the page over the network and the remote VM manager un-maps the page.

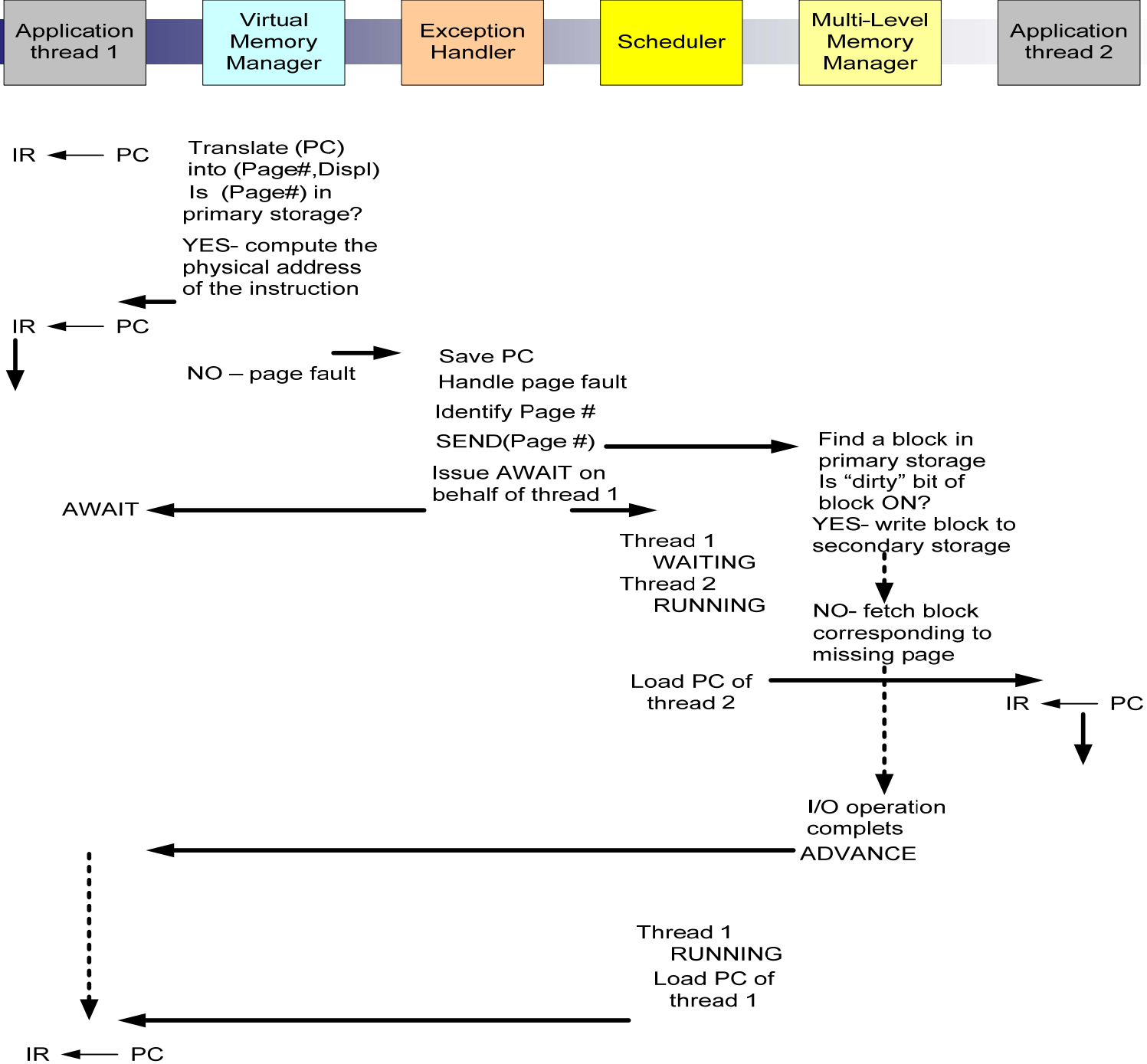
Multi-level memory management and virtual memory

- Two level memory system: RAM + disk.
  - Each page of an address space has an image in the disk
  - The RAM consists of blocks.
  - READ and WRITE from RAM → controlled by the VM manager
  - GET and PUT from disk → controlled by a multi-level memory manager
- Old design philosophy: integrate the two to reduce the instruction count
- New approach – modular organization
  - Implement the VM manager (VMM) in hardware. Translates virual addresses into physical addresses.
  - Implement the multi-level memory manager (MLMM) in the kernel in software. It transfers pages back and forth between RAM and the disk

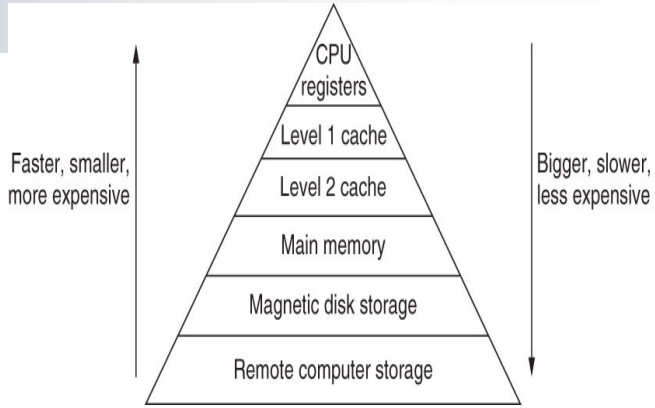
The modular design

- VM attempts to translate the virtual memory address to a physical memory address
- If the page is not in main memory VM generates a page-fault exception.
- The exception handler uses a SEND to send to an MLMM port the page number
- The SEND invokes ADVANCE which wakes up a thread of MLMM
- The MMLM invokes AWAIT on behalf of the thread interrupted due to the page fault.
- The AWAIT releases the processor to the SCHEDULER thread.




Application thread 1
Virtual Memory Manager
Exception Handler
Scheduler
Multi-Level Memory Manager
Application thread 2
IR ← PC
Translate (PC) into (Page#,Displ)
Is (Page#) in primary storage?
YES- compute the physical address of the instruction
IR ← PC
NO – page fault
Save PC
Handle page fault
Identify Page #
SEND(Page #)
Find a block in primary storage
Is "dirty" bit of block ON?
YES- write block to secondary storage
Issue AWAIT on behalf of thread 1
AWAIT
Thread 1 WAITING
Thread 2 RUNNING
NO- fetch block corresponding to missing page
Load PC of thread 2
IR ← PC
I/O operation complets
ADVANCE
Thread 1 RUNNING
Load PC of thread 1
IR ← PC

Name resolution in multi-level memories



- We consider pairs of layers:
  - Upper level of the pair → primary
  - Lower level of the pair → secondary
- The top level managed by the application which generates LOAD and STORE instructions to/from CPU registers from/to named memory locations
- The processor issues READs/WRITEs to named memory locations. The name goes to the primary memory device located on the same chip as the processor which searches the name space of the on-chip cache (L1 cache), the primary device with the L2 cache as secondary device.
- If the name is not found in *L1 cache* name space the Multi-Level Memory Manager (MLMM) looks at the *L2 cache* (off-chip cache) which becomes the primary with the main memory as secondary.
- If the name is not found in the L2 cache name space the MLMM looks at the main memory name space. Now the main memory is the primary device.
- If the name is not found in the main memory name space then the Virtual Memory Manager is invoked

The performance of a two level memory

- The latency $L_p << L_S$
  - L_P → latency of the primary device e.g., 10 nsec for RAM
  - L_S → latency of the secondary device, e.g., 10 msec for disk
- Hit ratio h→ the probability that a reference will be satisfied by the primary device.
- Average Latency (AS) ➔ AS = h x L_P + (1-h) L_S.
- Example:
  - L_P = 10 nsec (primary device is main memory)
  - L_S = 10 msec (secondary device is the disk)
  - Hit ratio h= 0.90 ➔ AS= 0.9 x 10 + 0.1 x 10,000,000 = 1,000,000.009 nsec~ 1000 microseconds = 1 msec
  - Hit ratio h= 0.99 ➔ AS= 0.99 x 10 + 0.01 x 10,000,000 = 100,000.0099 nsec~ 100 microseconds = 0.1 msec
  - Hit ratio h= 0.999 ➔ AS= 0.999 x 10 + 0.001 x 10,000,000 = 10,000.0099 nsec~ 10 microseconds = 0.01 msec
  - Hit ratio h= 0.9999 ➔ AS= 0.999 0x 10 + 0.001 x 10,000,000 = 1,009.99 nsec~ 1 microsecond

    This considerable slowdown is due to the very large discrepancy (six orders of magnitude) between the primary and the secondary device.

The performance of a two level memory (cont'd)

Statement: if each reference occurs with equal frequency to a cell in the primary and in the secondary device then *the combined memory will operate at the speed of the secondary device*.

- The size $Size_P << Size_S$ $Size_S = K \times Size_P$ with K large (1/K small)
  - $Size_P$ → number of cells of the primary device
  - $Size_S$ → number of cells of the secondary device

$$AverageLatency = \frac{Size_P}{Size_P + Size_S} L_P + \frac{Size_S}{Size_P + Size_S} L_S = \frac{1}{1+K} L_P + \frac{1}{1+1/K} L_S \approx L_S$$

Locality of reference

- Concentration of references
  - Spatial locality of reference
  - Temporal locality of reference
- Reasons for locality of references
  - Programs consists of sets of sequential instructions interrupted by branches
  - Data structures group together related data elements
- Working set → the collection of references made by an application in a given time window.
- If the working set is larger than the number of cells of the primary device significant performance degradation.

Memory management elements at each level

1. The string of references directed at that level.
2. The capacity at that level
3. The bring in policies
   1. On demand → bring the cell to the primary device from the secondary device when it is needed. E.g., demand paging
   2. Anticipatory. E.g. pre-paging
4. The replacement policies
   - FIFO → First in first out
   - OPTIMAL → what a clairvoyant multi-level memory manager would do. Alternatively, construct the string of references and use it for a second execution of the program (with the same data as input).
   - LRU – Least Recently Used → replace the page that has not been referenced for the longest time.
   - MSU – Most Recently Used → replace the page that was referenced most recently

Page replacement policies; Belady's anomaly

- In the following examples we use a given string of references to illustrate several page replacement policies.
- We consider a primary device (main memory) with a capacity of three or four blocks and a secondary device (the disk) where a replica of all pages reside.
- Once a block has the "dirty bit" on it means that the page residing in that block was modifies and must be written back to the secondary device before being replaced.
- The capacity of the primary device is important. One expects that increasing the capacity, in our case the number of blocs in RAM leads to a higher hit ratio. That is not always the case as our examples will show. This is the Belady's anomaly.
- Note: different results are obtained with a different string of references!!

FIFO Page replacement algorithm→ PS: Primary storage

Time intervals	1	2	3	4	5	6	7	8	9	10	11	12	Total number of page faults
Reference string	0	1	2	3	0	1	4	0	1	2	3	4	
Block 1 in PS	-	0	0	0	3	3	3	4	4	4	4	4	
Block 2 in PS	-	-	1	1	1	0	0	0	0	0	2	2	
Block 3 in PS	-	-	-	2	2	2	1	1	1	1	1	3	
Page OUT	-	-	-	0	1	2	3	-	-	0	1	-	
Page IN	0	1	2	3	0	1	4	-	-	2	3	-	9

Block 1 in PS	-	0	0	0	0	0	0	4	4	4	4	3	
Block 2 in PS	-	-	1	1	1	1	1	1	0	0	0	0	
Bloch 3 in PS	-	-	-	2	2	2	2	2	2	1	1	1	
Block 4 in PS	-	-	-	-	3	3	3	3	3	3	2	2	
Page OUT	-	-	-	-	-	-	0	1	2	3	4	0	
Page IN	0	1	2	3	-	-	4	0	1	2	3	4	10

OPTIMAL page replacement algorithm

Time intervals	1	2	3	4	5	6	7	8	9	10	11	12	Total number of page faults
Reference string	0	1	2	3	0	1	4	0	1	2	3	4	
Block 1 in PS	-	0	0	0	0	0	0	0	0	0	2	3	
Block 2 in PS	-	-	1	1	1	1	1	1	1	1	1	1	
Block 3 in PS	-	-	-	2	3	3	3	4	4	4	4	4	
Page OUT	-	-	-	2	-	-	3	-	-	0	2	-	
Page IN	0	1	2	3	-	-	4	-	-	2	3	-	7

Block 1 in PS	-	0	0	0	0	0	0	0	0	0	0	3	
Block 2 in PS	-	-	1	1	1	1	1	1	1	1	1	1	
Bloch 3 in PS	-	-	-	2	2	2	2	2	2	2	2	2	
Block 4 in PS	-	-	-	-	3	3	3	4	4	4	4	4	
Page OUT	-	-	-	-	-	-	3	-	-	-	0	-	
Page IN	0	1	2	3	-	-	4	-	-	-	3	-	6

LRU page replacement algorithm

Time intervals	1	2	3	4	5	6	7	8	9	10	11	12	Total number of page faults
Reference string	0	1	2	3	0	1	4	0	1	2	3	4	
Block 1 in PS	-	0	0	0	0	0	0	0	0	0	0	3	
Block 2 in PS	-	-	1	1	2	1	1	1	1	1	1	1	
Block 3 in PS	-	-	-	2	3	3	3	4	4	4	2	2	
Page OUT	-	-	-	1	-	2	3	-	-	4	0	1	
Page IN	0	1	2	3	-	1	4	-	-	2	3	4	9

Block 1 in PS	-	0	0	0	0	0	0	0	0	0	0	0	
Block 2 in PS	-	-	1	1	1	1	1	1	1	1	1	1	
Bloch 3 in PS	-	-	-	2	2	2	2	4	2	2	2	2	
Block 4 in PS	-	-	-	-	3	3	3	3	4	4	4	3	
Page OUT	-	-	-	-	-	-	2	-	-	-	4	0	
Page IN	0	1	2	3	-	-	4	-	-	-	3	4	7

LRU, OPTIMAL, MRU

- LRU looks only at history
- OPTIMAL “knows” not only the history but also the future.
- In some particular cases Most Recently Used Algorithm performs better than LRU.
- Example: primary device with 4 cells.

Reference string	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
LRU	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
MRU	F	F	F	F	-	-	-	-	F	-	-	-	F	-	-

The OPTIMAL replacement policy keeps in the 3-blocks primary memory the same pages as it does in case of the 4-block primary memory.

Time intervals	1	2	3	4	5	6	7	8	9	10	11	12	Total number of page faults
Reference string	0	1	2	3	0	1	4	0	1	2	3	4	
Block 1 in PS	-	0	0	0	0	0	0	0	0	0	2	3	
Block 2 in PS	-	-	1	1	1	1	1	1	1	1	1	1	
Block 3 in PS	-	-	-	2	3	3	3	4	4	4	4	4	
Page OUT	-	-	-	2	-	-	3	-	-	0	2	-	
Page IN	0	1	2	3	-	-	4	-	-	2	3	-	7

Block 1 in PS	-	0	0	0	0	0	0	0	0	0	0	3	
Block 2 in PS	-	-	1	1	1	1	1	1	1	1	1	1	
Bloch 3 in PS	-	-	-	2	2	2	2	2	2	2	2	2	
Block 4 in PS	-	-	-	-	3	3	3	4	4	4	4	4	
Page OUT	-	-	-	-	-	-	3	-	-	-	0	-	
Page IN	0	1	2	3	-	-	4	-	-	-	3	-	6

The FIFO replacement policy does not keep in the 3-blocks primary memory the same pages as it does in case of the 4-block primary memory.

Time intervals	1	2	3	4	5	6	7	8	9	10	11	12	Total number of page faults
Reference string	0	1	2	3	0	1	4	0	1	2	3	4	
Block 1 in PS	-	0	0	0	3	3	3	4	4	4	4	4	
Block 2 in PS	-	-	1	1	1	0	0	0	0	0	2	2	
Block 3 in PS	-	-	-	2	2	2	1	1	1	1	1	3	
Page OUT	-	-	-	0	1	2	3	-	-	0	1	-	
Page IN	0	1	2	3	0	1	4	-	-	2	3	-	9

Block 1 in PS	-	0	0	0	0	0	0	4	4	4	4	3	
Block 2 in PS	-	-	1	1	1	1	1	1	0	0	0	0	
Bloch 3 in PS	-	-	-	2	2	2	2	2	2	1	1	1	
Block 4 in PS	-	-	-	-	3	3	3	3	3	3	2	2	
Page OUT	-	-	-	-	-	-	0	1	2	3	4	0	
Page IN	0	1	2	3	-	-	4	0	1	2	3	4	10

The LRU replacement policy keeps in the 3-blocks primary memory the same pages as it does in case of the 4-block primary memory.

Time intervals	1	2	3	4	5	6	7	8	9	10	11	12	Total number of page faults
Reference string	0	1	2	3	0	1	4	0	1	2	3	4	
Block 1 in PS	-	0	0	0	0	0	0	0	0	0	0	3	
Block 2 in PS	-	-	1	1	2	1	1	1	1	1	1	1	
Block 3 in PS	-	-	-	2	3	3	3	4	4	4	2	2	
Page OUT	-	-	-	2	-	-	3	-	-	0	2	-	
Page IN	0	1	2	3	-	1	4	-	-	2	3	4	9

Block 1 in PS	-	0	0	0	0	0	0	0	0	0	0	0	
Block 2 in PS	-	-	1	1	1	1	1	1	1	1	1	1	
Bloch 3 in PS	-	-	-	2	2	2	2	4	2	2	2	2	
Block 4 in PS	-	-	-	-	3	3	3	3	4	4	4	3	
Page OUT	-	-	-	-	-	-	2	-	-	-	4	0	
Page IN	0	1	2	3	-	-	4	-	-	-	3	4	7

The FIFO replacement policy does not keep in the 3-blocks primary memory the same pages as it does in case of the 4-block primary memory

Time intervals	1	2	3	4	5	6	7	8	9	10	11	12	Total number of page faults
Reference string	0	1	2	3	0	1	4	0	1	2	3	4	
Block 1 in PS	-	0	0	0	3	3	3	4	4	4	4	4	
Block 2 in PS	-	-	1	1	1	0	0	0	0	0	2	2	
Block 3 in PS	-	-	-	2	2	2	1	1	1	1	1	3	
Page OUT	-	-	-	0	1	2	3	-	-	0	1	-	
Page IN	0	1	2	3	0	1	4	-	-	2	3	-	9

Block 1 in PS	-	0	0	0	0	0	0	4	4	4	4	3	
Block 2 in PS	-	-	1	1	1	1	1	1	0	0	0	0	
Bloch 3 in PS	-	-	-	2	2	2	2	2	2	1	1	1	
Block 4 in PS	-	-	-	-	3	3	3	3	3	3	2	2	
Page OUT	-	-	-	-	-	-	0	1	2	3	4	0	
Page IN	0	1	2	3	-	-	4	0	1	2	3	4	10

How to avoid Belady's anomaly

- The OPTIMAL and the LRU algorithms have the subset property, a primary device with a smaller capacity hold a subset of the pages a primary device with a larger capacity could hold.
- The subset property creates a total ordering. If the primary system has 1 blocks and contains page A a system with two block add page B, and a system with three blocks will add page C. Thus we have a total ordering

  A→B → C or (A,B,C)
- Replacement algorithms that have the subset property are called "stack" algorithms.
- If we use stack replacement algorithms a device with a larger capacity can never have more page faults than the one with a smaller capacity.

  m→ the pages held by a primary device with smaller capacity

  n → the pages held by a primary device with larger capacity

  m is a subset of n

Simulation analysis of page replacement algorithms

- Given a reference string *we can carry out the simulation for all possible cases when the capacity of the primary storage device varies from 1 to n with a single pass.*
- At each new reference the some page move to the top of the ordering and the pages that were above it either move down or stay in the same place as dictated by the replacement policy. We record whether this movement correspond to paging out, movement to the secondary storage.

Simulation of LRU page replacement algorithm

Time	1	2	3	4	5	6	7	8	9	10	11	12	
Reference string	**0**	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**	

Stack contents after refrence													Total number of page faults
	0	1	2	3	0	1	4	0	1	2	3	4	
	-	0	1	2	3	0	1	4	0	1	2	3	
	-	-	0	2	2	3	0	1	4	0	1	2	
	-	-	-	0	1	2	3	3	3	4	0	1	
	-	-	-	-	-	-	2	2	2	5	4	0	

Size 1 in/out	**0/-**	**1/0**	**2/1**	**3/2**	**0/3**	**1/0**	**4/1**	**0/4**	**1/0**	**2/1**	**3/2**	**4/3**	**12**
Size 2 in/out	**0/-**	**1/-**	**2/0**	**3/1**	**0/2**	**1/3**	**4/0**	**0/1**	**1/4**	**2/0**	**3/1**	**4/2**	**12**
Size 3 in/out	**0/-**	**1/-**	**2/-**	**3/0**	**0/1**	**1/2**	**4/3**	**-/-**	**-/-**	**2/4**	**3/0**	**4/1**	**10**
Size 4 in/out	**0/-**	**1/-**	**2/-**	**3/-**	**-/-**	**-/-**	**4/2**	**-/-**	**-/-**	**2/3**	**3/4**	**4/0**	**8**
Size 5 in/out	**0/-**	**1/-**	**2/-**	**3/-**	**-/-**	**-/-**	**4/-**	**-/-**	**-/-**	**-/-**	**-/-**	**-/-**	**5**

Simulation of OPTIMUM

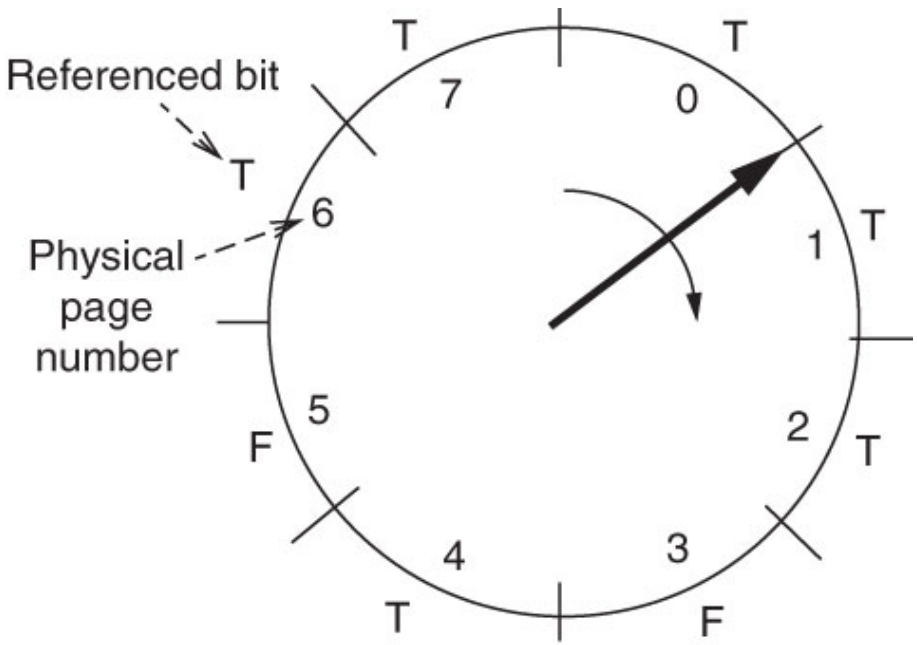
Time	1	2	3	4	5	6	7	8	9	10	11	12	
Reference string	**0**	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**	

Stack contents after refrence	0	1	2	3	0	1	4	0	1	2	3	4	Total number of page faults
	-	0	0	0	3	0	0	4	0	0	0	0	
	-	-	1	1	1	3	1	1	4	4	4	3	
	-	-	-	2	2	2	3	3	3	3	2	2	
	-	-	-	-	-	-	2	2	2	1	1	1	

Size 1 victim	-	**0**	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**11**
Size 2 victim	-	-	**1**	**2**	-	**3**	**1**	-	**1**	**2**	**3**	**4**	**10**
Size 3 victim	-	-	-	**2**	-	-	**4**	-	-	**2**	**3**	-	**7**
Size 4 victim	-	-	-	-	-	-	**4**	-	-	**2**	-	-	**6**
Size 5 victim	-	-	-	-	-	-	-	-	-	-	-	-	**5**

Clock replacement algorithm

- Approximates LRU with a minimum
  - Additional hardware: one reference bit for each page
  - Overhead
- Algorithm activated :
  - when a new page must be brought in move the pointer of a virtual clock in clockwise direction
  - if the arm points to a block with reference bit TRUE
    - Set it FALSE
    - Move to the next block
  - if the arm points to a block with reference bit FALSE
    - The page in that block could be removed (has not been referenced for a while)
    - Write it back to the secondary storage if the “dirty” bit is on (if the page has been modified.


Referenced bit
T
Physical page number
6
7
T
0
T
1
T
2
T
3
F
4
T
5
F