COMPS267F Chapter 8

Virtual Memory

Dr. Andrew Kwok-Fai LUI

Aim of the Chapter

- Describe the operation of virtual memory system
- Explain how memory virtualization improve main memory utilization

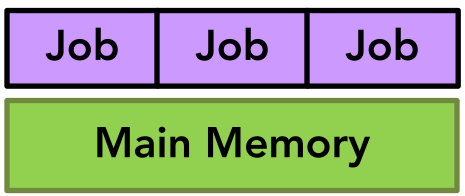
Main Memory Utilization

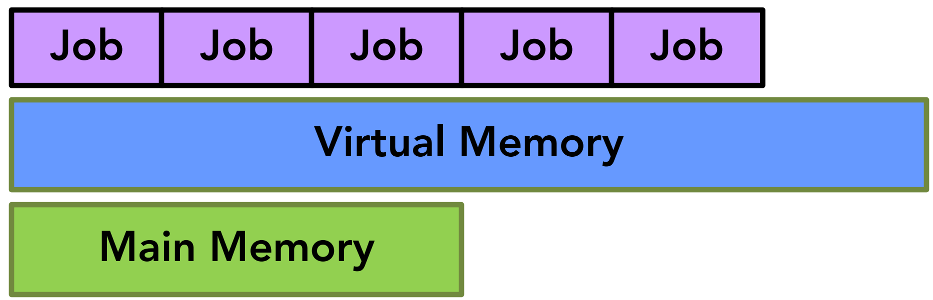
- Factors of utilization of main memory
  - The number of loaded processes
    - Level of multi-programming
  - Fragmentation
    - Techniques to minimize memory wastage
- Achieving 100% is ideal, what about 200%?
  - A computer with 4G on board main memory can offer 8G main memory for program execution?

Virtualization of Memory

Conventional Main Memory System

Virtual Memory System





All physical memory is utilized, cannot load more process

Placing a layer of virtual memory on top of physical memory, the larger size virtual memory allows more jobs to be loaded

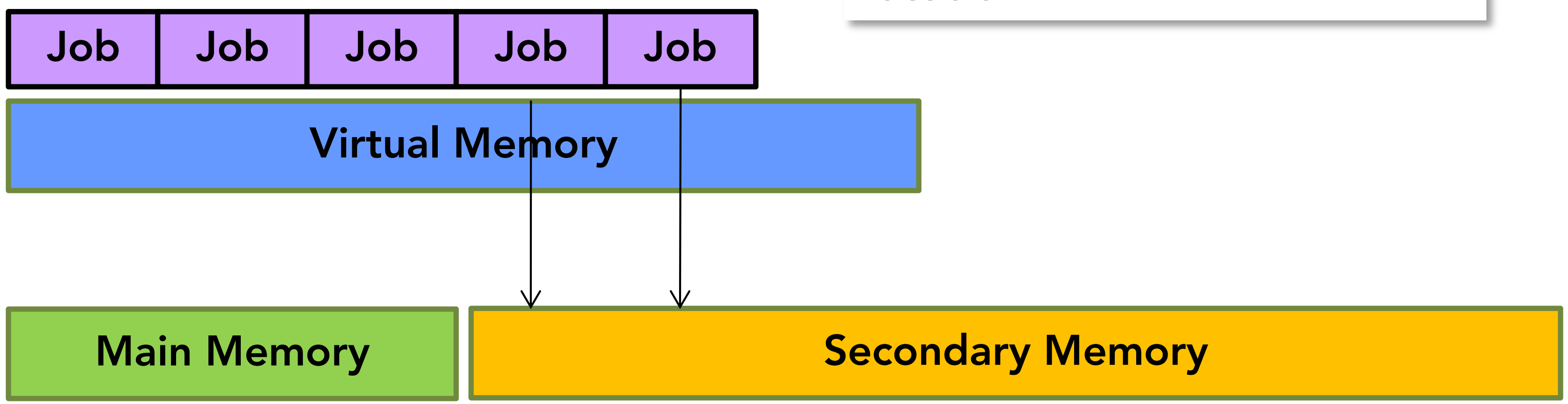
Virtualization of Memory

- Added a layer of abstraction
  - The layer provides more memory than the actual physical memory
    - Allowing more processes to be loaded
  - Example: 4GB physical memory can become 8GB virtual memory to allow a maximum of 8 1G processes to be loaded
- How can it be done?
  - Making use of secondary memory
  - Clever movement of data between main memory and secondary memory

Virtualization of Memory

Virtual Memory System

The virtual memory layer hides away the actual storage location



Some of the jobs are actually not loaded in the main memory. They are stored in the hard disk

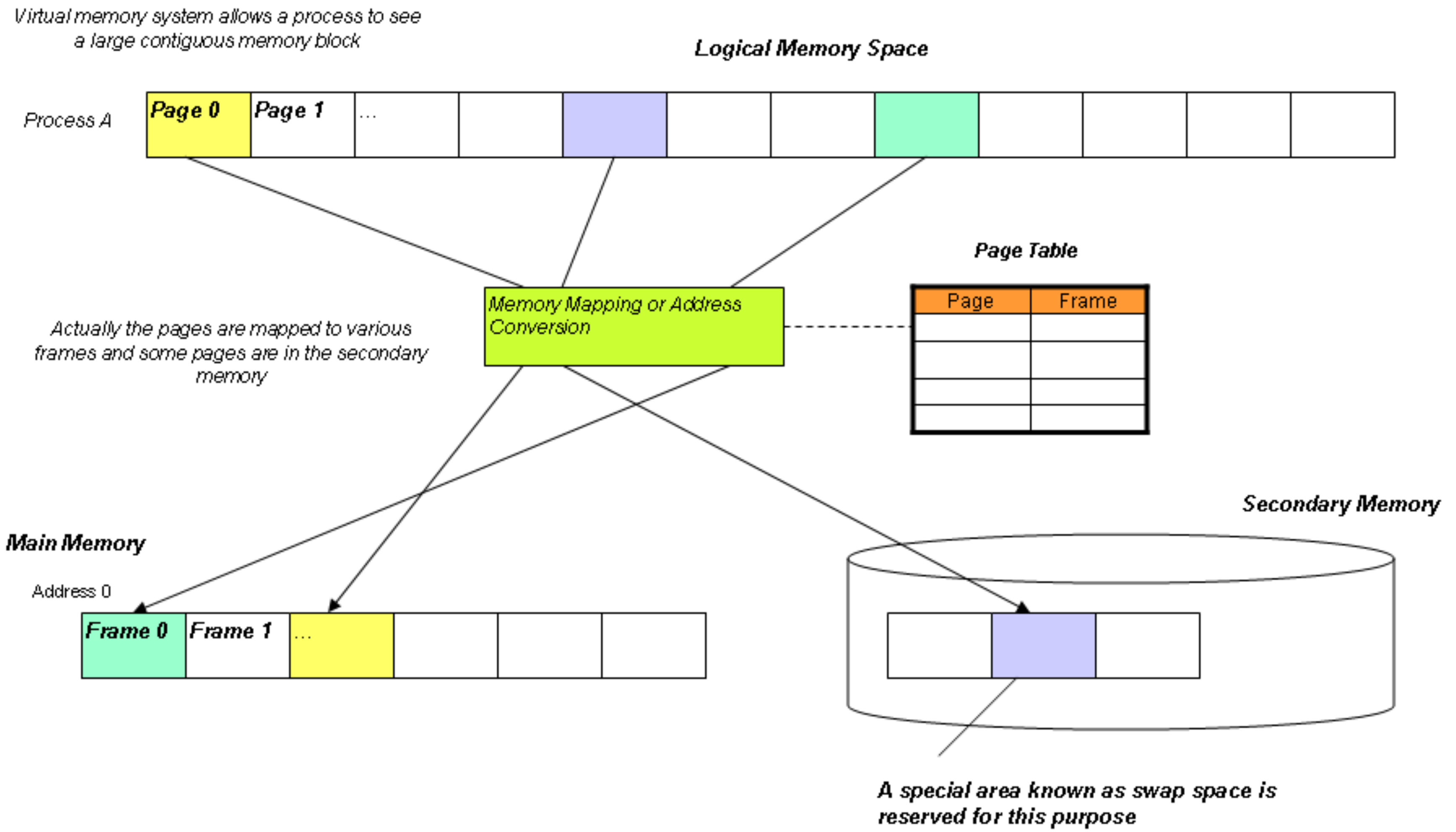
波坦金村（Potemkin Village）(Credits: 世界真的很大）

virtual memory

Virtualization of Memory

- Illusion created by the virtual memory layer
  - A process' memory block is in one contiguous piece logically
    - The process may be split in several physical memory blocks
  - A process' memory block is all loaded in the physical memory
    - Some memory blocks of a process are not in the physical memory
    - They are stored in hard disk.
  - Every memory address of a process is directly accessible
    - Some memory address of a process is actually a location in the hard disk
- The processes are feeling good and they cannot see the reality.

Virtualization of Memory



Virtualization of Memory

- Virtual address space is the logical view of a program stored in a large contiguous block of memory
  - In reality the program would be stored in multiple separated blocks in physical memory.
  - Operations of a virtual memory system
    - Page table look-up
    - Dynamic binding
    - Dynamic loading
    - Swapping-in and Swapping-out

Partial Loading of Programs

- Partial loading of programs is an important feature in virtual memory systems
- Partial loading of programs is clever
  - Only a fraction of the functions of a program is needed
  - Many branches in a program deal with error handling
    - Rarely executed
  - Data structures such as arrays are often declared with a size larger than needed
    - Some parts of arrays are never needed

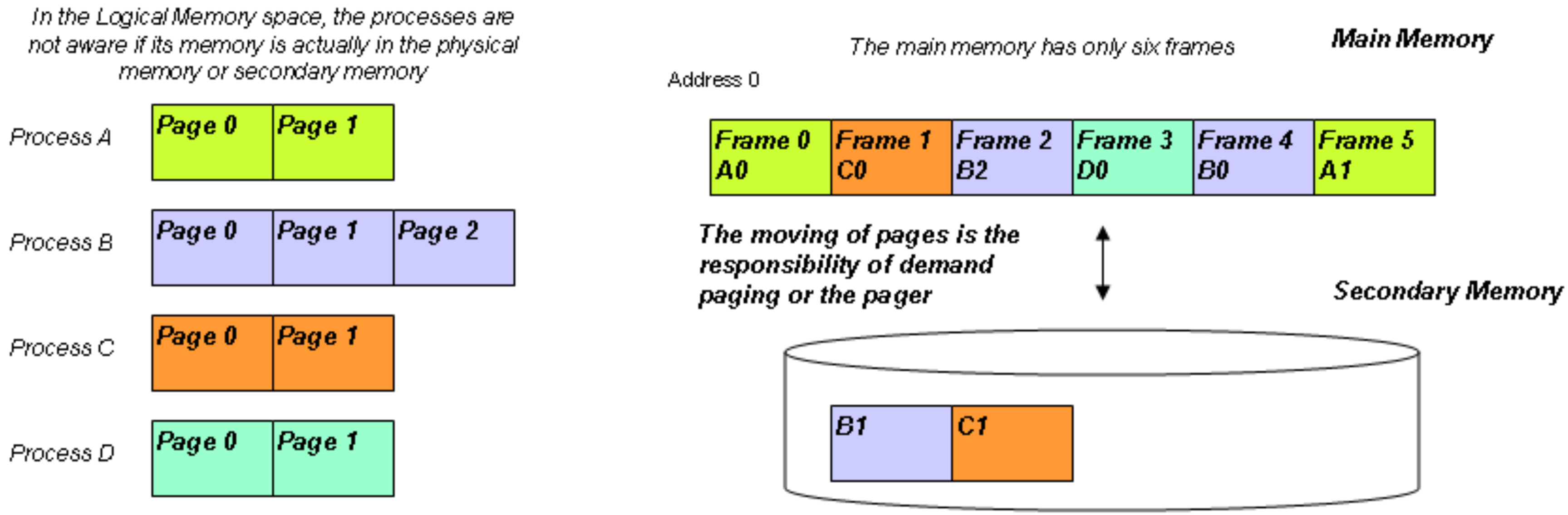
Advantages of Partial Loading of Programs

- The size of the logical memory space of a process can now be greater than the physical memory size
  - 1GB memory can allow a program declaring array of 2GB size
- More programs (or processes) can be executed at the same time.
  - Increases the level of multi-programming
- The program start-up time is faster because of the less loading time needed
  - Only the needed parts of the program are loaded first

Illusion of Virtual Memory Systems

- Virtual memory system creates a memory illusion
  - To a program the memory space is large and contiguous
  - Hides away that some pages are in different parts in the main memory and other pages are even saved in the backing store in secondary memory

Illusion of Virtual Memory Systems



demand paging

Demand Paging

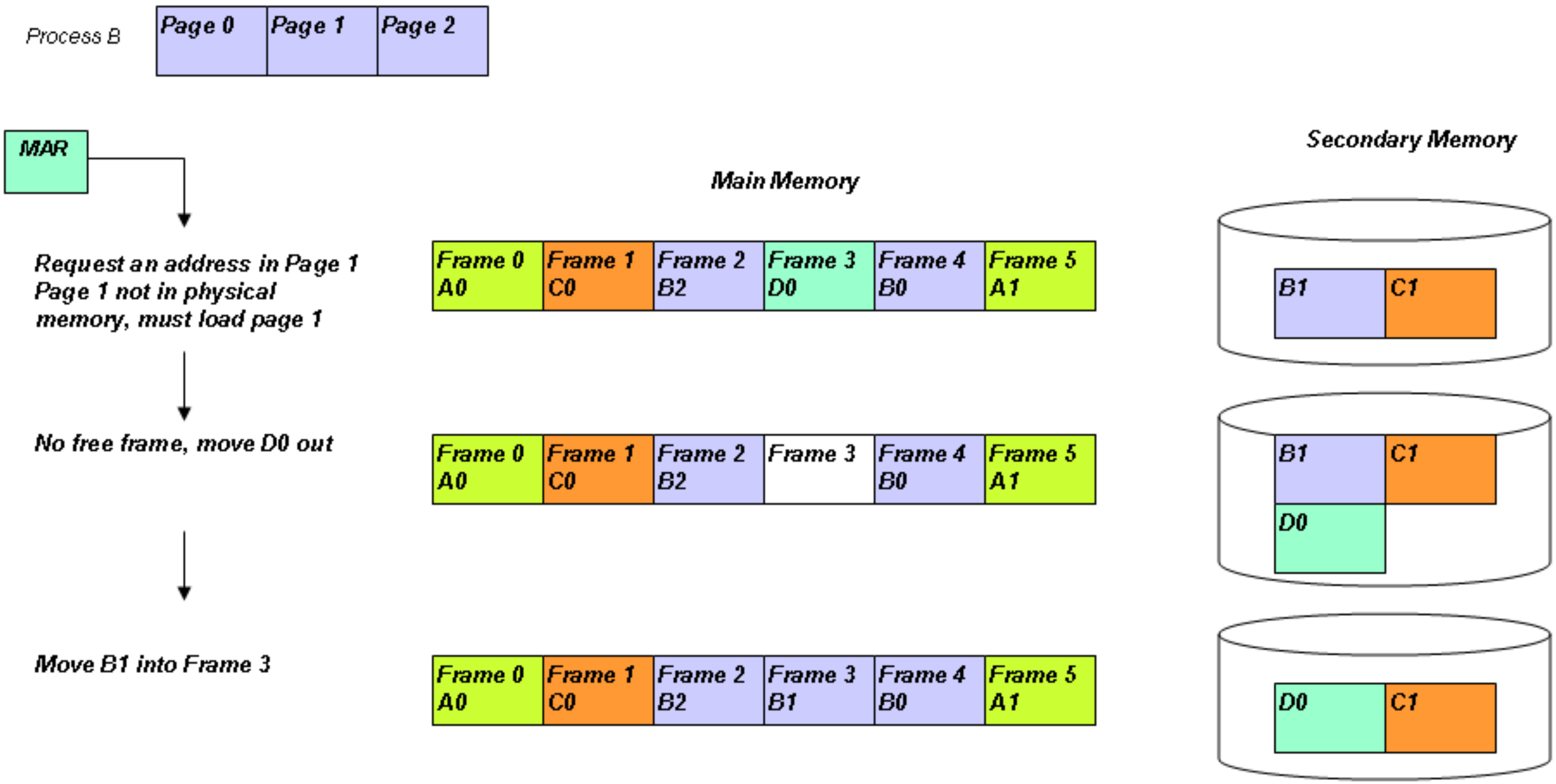
- Virtual memory is implemented by a demand-paging system
  - Pages are loaded when they are actually needed
    - Another version of swapping-in
  - A page not needed will be selected and moved out by a pager
    - Another version of swapping-out
    - A component called pager selects and moves the page out
- On-demand to the need of processes

Demand Paging

- The pager will ideally move in the pages that will be needed in the near future
- Page moved in to the main memory, but soon the whole process is swapped out
  - increase the demand on the main memory and IO unnecessarily
  - Difficult to predict

Demand Paging

Consider that Process B is in Running state and being executed by the CPU



Pure Demand Paging

- Going to the extreme in demand paging
- Never loads in a page until it is needed,
  - Including the very first page
  - A special case in which a process in the Ready state is not loaded into the main memory at all

Demand Paging

- Demand paging moves pages between the main memory and the secondary memory
- The hard-disk has a space allocated for this purpose
  - Swap space
  - Outside the file system
  - Disk access to the swap space is generally faster than access the file systems

Page Tables in Demand Paging

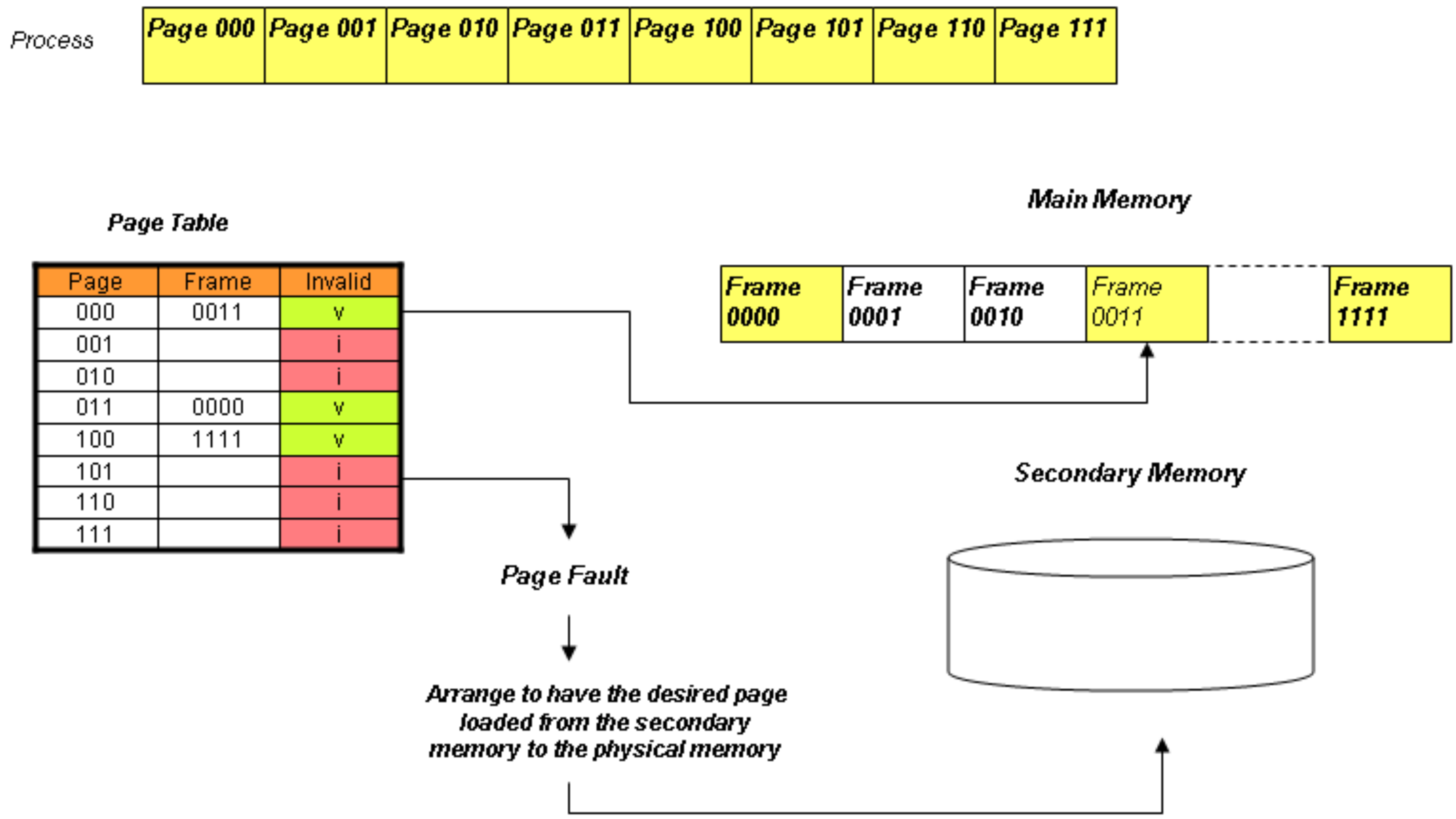
- The page table needs augmentation to support memory operation
  - Each page is stored either in main memory or hard disk
  - Valid-invalid bit to indicate the status of a page
  - Valid page means the memory address is in main memory
  - Access to an invalid page would cause a **page fault**

Page Fault

- The handling of a page fault involves a number of steps
  - Kernel checks true page fault or illegal memory access
  - If it is the latter, then the process is terminated
  - A free frame in the physical memory is found (from **frame table** or free-frame list) and load in the page to the free frame
  - Page replacement if no free frame
  - IO completed, update page table, resume instruction

Page Fault

Consider that Process B is in Running state and being executed by the CPU



Page	Frame	Invalid
000	0011	v
001		i
010		i
011	0000	v
100	1111	v
101		i
110		i
111		i

Page Fault

- Rate of page fault in actual operations is fortunately not too high
- Page fault is a costly situation
  - Memory access time is in the tens of nanosecond range
  - Disk access time is in the millisecond range
  - A difference in the scale of hundreds of thousands

Page Fault

- Multiple page faults per instruction are possible if a page fault occurs to the access to the instruction and the data
  - This would cause a very poor system performance
- Fortunately we have locality of references
  - Predict which page is needed in the near future
  - Keeping the page fault frequency low

Example: Page Fault

Example: Effective Memory Access Time in Demand Paging System

The effective memory access time (EMAT) is the average amount of time taken to perform a memory operation.

If p is the probability of page fault, the effective access time is given in the following.

EMAT = (1 – p) * time (memory access) + p * time (page fault handling)

In handling a memory operation, there are two possibilities:

1. Page fault not occurred: the time taken is time (memory access).
2. Page fault occurred: the time taken is time (page fault handling).

Normally the page fault handling time is several orders of magnitude higher than memory access time.

Assuming that the memory access time is 100 nanoseconds, and the page-fault service time is 25 milliseconds. The page fault rate is one in a thousand. Evaluate the effective memory access time.

Example: Page Fault

Example: Effective Memory Access Time in Demand Paging System

The effective memory access time (EMAT) is the average amount of time taken to perform a memory operation.

If p is the probability of page fault, the effective access time is given in the following.

EMAT = (1 – p) * time (memory access) + p * time (page fault handling)

In handling a memory operation, there are two possibilities:

1. Page fault not occurred: the time taken is time (memory access).
2. Page fault occurred: the time taken is time (page fault handling).

Normally the page fault handling time is several orders of magnitude higher than memory access time.

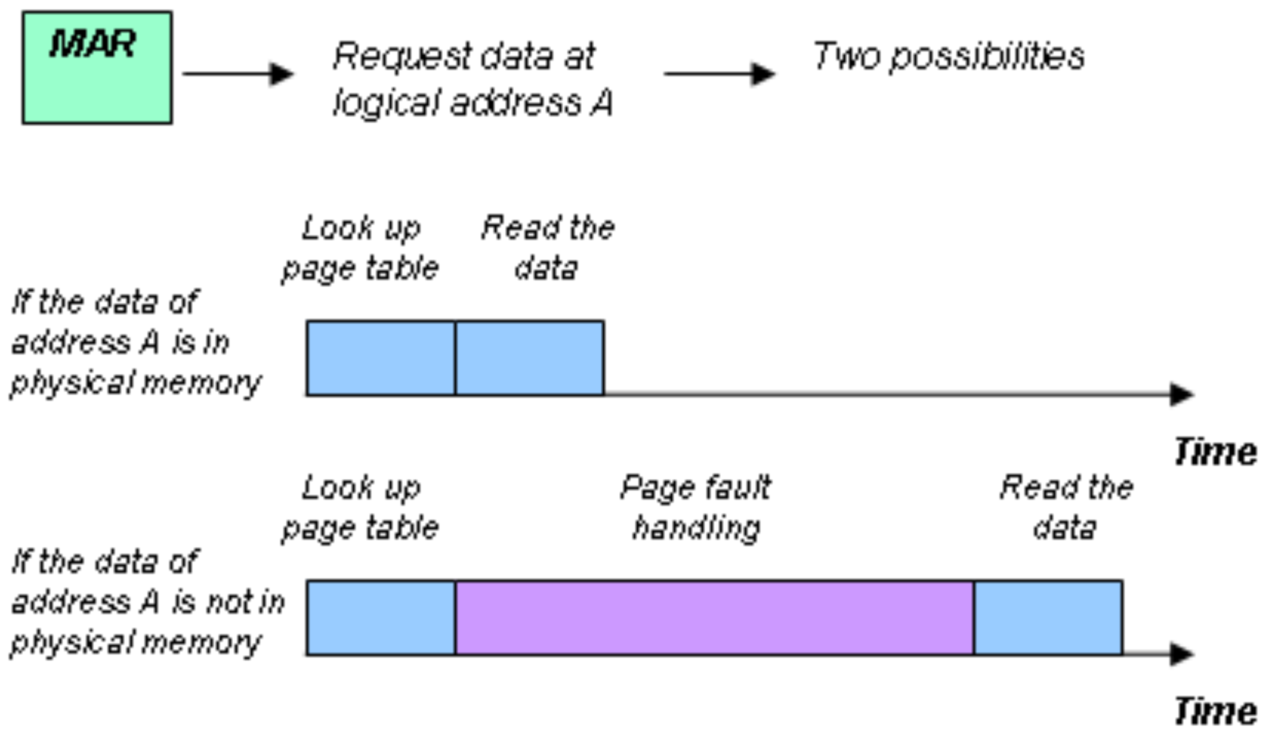
Assuming that the memory access time is 100 nanoseconds, and the page-fault service time is 25 milliseconds. The page fault rate is one in a thousand. Evaluate the effective memory access time.

> Assume that the page table is stored in the main memory.
>
> EMAT = (1- 0.001) * (100 ns + 100 ns) + (0.001) * (25000000 + 100 + 100) ns
>
> = 25200 ns

Is the resulting effective access time acceptable?

> This is not acceptable. In the ideal case of no page fault, the EMAT would be 200 ns. So there is 12500% increase due to page fault handling.

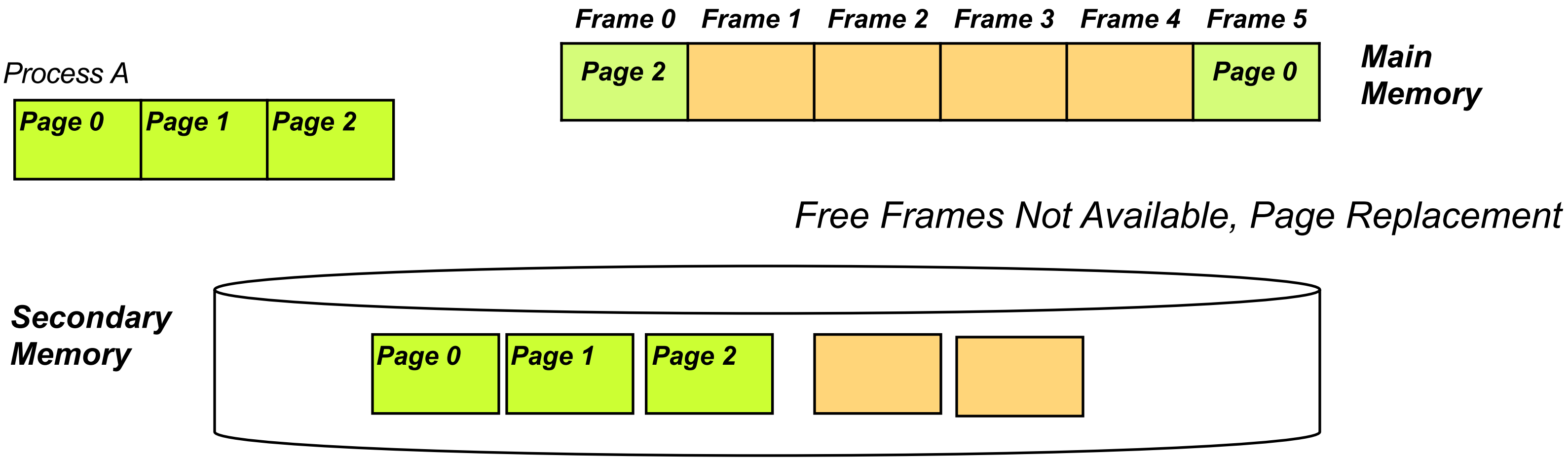
Example: Page Fault

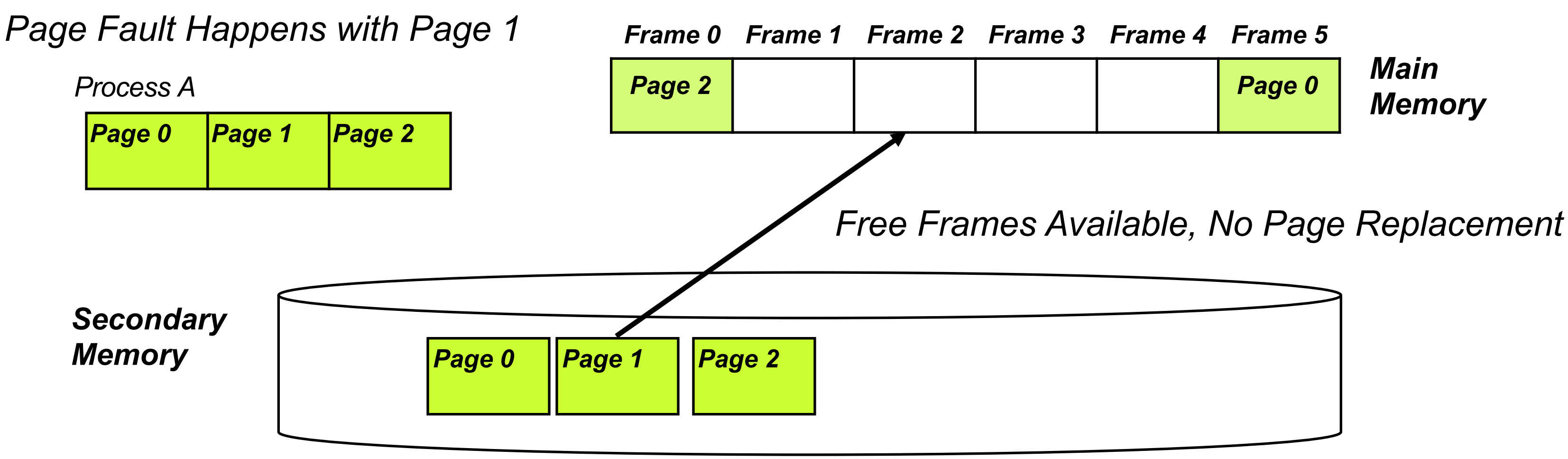


Summary

- Virtual memory system makes main memory larger
  - Some pages are not loaded into the main memory
    - Stored in the secondary memory instead
- Page fault
  - Occurs when the required page is not in the main memory
    - In the page table the page has an invalid bit set on
- Demand paging
  - Load the page from secondary memory to main memory when page fault occurs
    - Page fault handling
    - Time consuming process

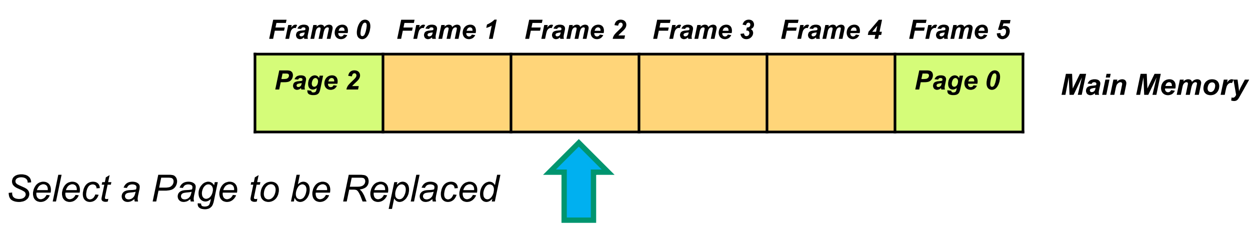
page replacement


Process A
Page 0
Page 1
Page 2
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Page 2
Page 0
Main Memory
Free Frames Not Available, Page Replacement
Secondary Memory
Page 0
Page 1
Page 2


Page Fault Happens with Page 1
Process A
Page 0
Page 1
Page 2
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Page 2
Page 0
Main Memory
Free Frames Available, No Page Replacement
Secondary Memory
Page 0
Page 1
Page 2

Page Replacement

- A page fault occurs with a user program
  - Kernel checks the page table and address that this is due to a real page fault.
  - The page is found to be on the hard disk
  - Check if free frame is available
- No free frame is available then <u>page replacement</u> is required.
  - Select a page to be replaced



Page Replacement

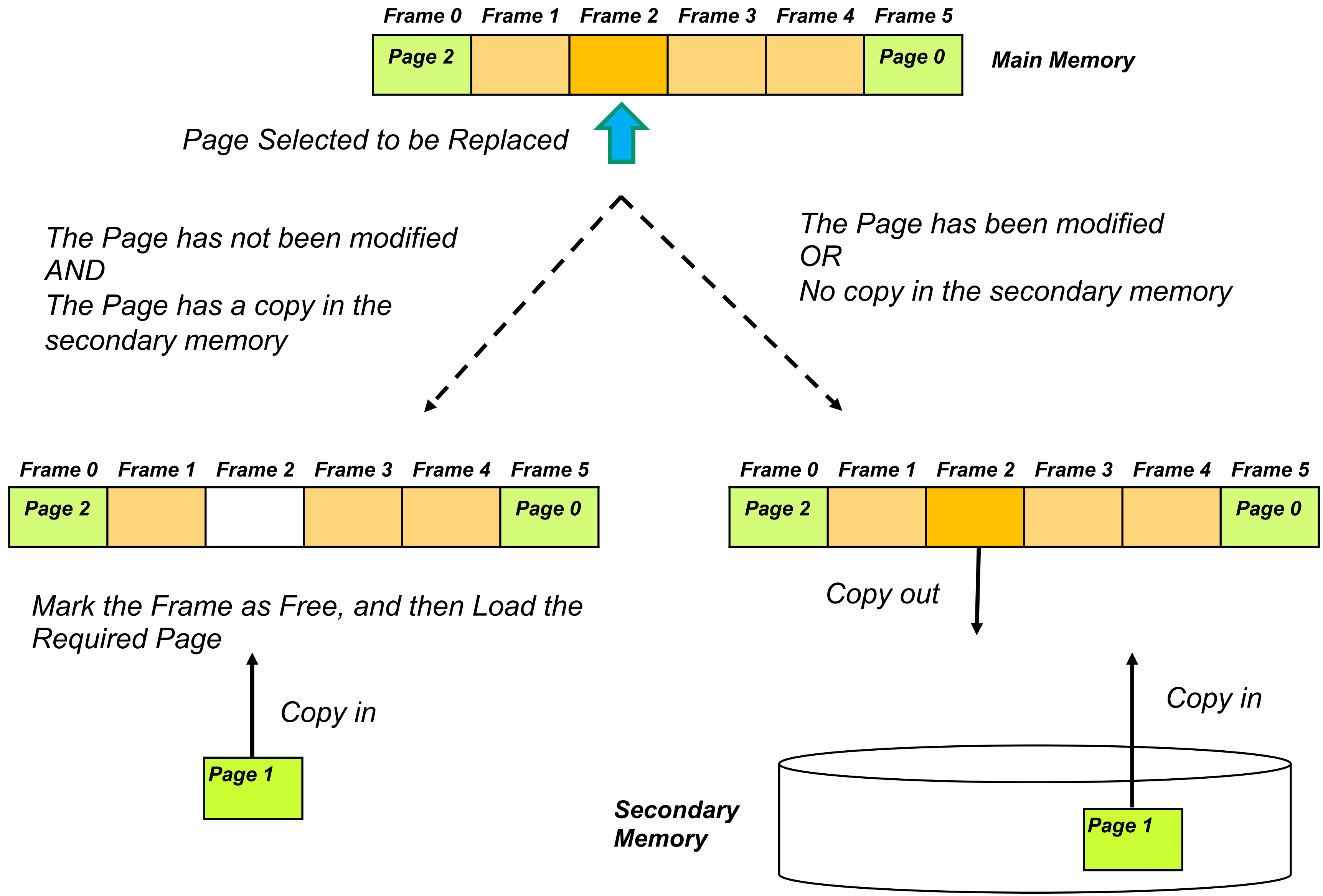
- A page replacement may cause two-page transfers (one in and one out)
  - Effectively page fault service time may be doubled
- A copy of the page in the hard disk may be useful
  - The copying of the outgoing page is required only if the page has been changed
  - A modify bit (or a dirty bit) is in each page table entry
  - Recording whether a page has been modified
- The pager can examine the modify bit to decide whether the outgoing copying is required

Page Replacement

- All frames in the physical memory are occupied and a new frame is required
  - Page replacement is not always needed
    - When free frames are available
  - Page replacement is needed to make available memory space for the new frame
  - A consequence of over-allocation of memory

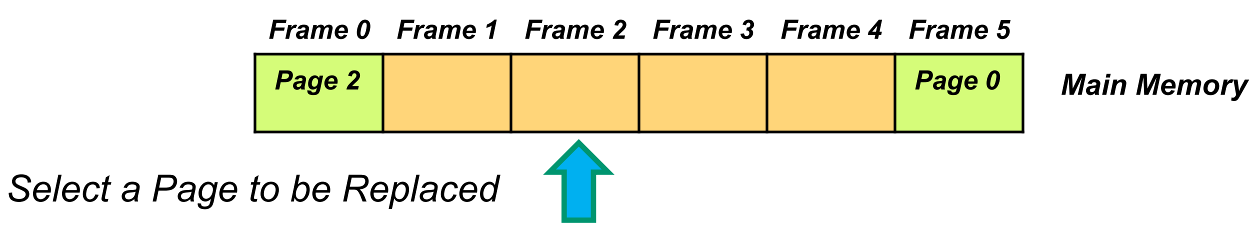
Steps of Page Replacement

- Identify the desired page which is stored on hard disk
- Find a free frame from the frame table
  - Use a free frame is exist
  - Use page replacement algorithm to swap out an existing frame
- The modify bit of the selected frame is examined to see if copying out is really required
- When the selected frame is made free, the desired page is loaded in
- The data structures (page tables, frame table) are updated, and the process is restarted


Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Page 2
Page 0
Main Memory
Page Selected to be Replaced
The Page has not been modified
AND
The Page has a copy in the
secondary memory
The Page has been modified
OR
No copy in the secondary memory
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Page 2
Page 0
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Page 2
Page 0
Mark the Frame as Free, and then Load the
Required Page
Copy out
Copy in
Page 1
Copy in
Secondary
Memory
Page 1

Page Replacement Algorithms

- There are ways of selecting which page in the physical memory is to be replaced
- The different strategies adopted are known as page replacement algorithms
  - OS designers to choose one giving the lowest page fault rate



Page Replacement Algorithms

- Algorithms to discuss
  - First-in-first-out (FIFO) (Baseline)
  - Optimal (OPT)
  - Least-recently-used (LRU)
  - Least-frequently-used (LFU)
- The aim is to reduce page fault rate
  - The baseline provides a performance level for comparison

Page Replacement Algorithm: FIFO

- FIFO selects the page that has been in memory for the longest time for removal
  - Based on the assumption that this page is less likely to be used
- Advantage of this algorithm
  - Simple to implement with a FIFO queue is set up to hold all pages in memory
  - When a page is brought into memory, it is inserted at the end of the queue

Page Replacement Algorithm: FIFO

Reference String 1 2 3 4 1 2 5 1 2 3 4 5

The reference string describes the order of PAGEID requested by a process

FIFO Page Replacement

Physical Memory

		1	2	3	4	1	2	5	1	2	3	4	5
Frame 0		1	1	1	4	4	4	5	5	5	5	5	5
Frame 1			2	2	2	1	1	1	1	1	3	3	3
Frame 2				3	3	3	2	2	2	2	2	4	4
		F	F	F	F	F	F	F			F	F	

Page Replacement Algorithm: FIFO

- The performance of this algorithm is questionable
  - Ignores the usage pattern of the pages
  - Locality of references
- Normally we expect that the performance improve with the amount of physical memory use
  - Belady's anomaly
  - Increasing the amount of physical memory use may worsen the performance of FIFO

Page Replacement Algorithm: FIFO

Reference String **1 2 3 4 1 2 5 1 2 3 4 5**

The reference string describes the order of PAGEID requested by a process

FIFO Page Replacement

Physical Memory (3 Frames)

		1	2	3	4	1	2	5	1	2	3	4	5
Frame 0		1	1	1	4	4	4	5	5	5	5	5	5
Frame 1			2	2	2	1	1	1	1	1	3	3	3
Frame 2				3	3	3	2	2	2	2	2	4	4
		F	F	F	F	F	F	F			F	F	

FIFO Page Replacement

Physical Memory (4 Frames)

		1	2	3	4	1	2	5	1	2	3	4	5
Frame 0		1	1	1	1	1	1	5	5	5	5	4	4
Frame 1			2	2	2	2	2	2	1	1	1	1	5
Frame 2				3	3	3	3	3	3	2	2	2	2
Frame 3					4	4	4	4	4	4	3	3	3
		F	F	F	F			F	F	F	F	F	F

Page Replacement Algorithm: OPT

- An algorithm for analysis only
  - Guaranteed to give the lowest page fault rate
  - OPT requires the system to know the exact page identity and timing of future paging demands
    - Replaces the page that will not be used for the longest period of time in the future
    - Impossible to know the future
    - Not practical
  - Provide the best-case performance (baseline)

Page Replacement Algorithm: OPT

Reference String

1 2 3 4 1 2 5 1 2 3 4 5

The reference string describes the order of PAGEID requested by a process

Physical Memory

OPT Page Replacement

		1	2	3	4	1	2	5	1	2	3	4	5
Frame 0		1	1	1	1	1	1	1	1	1	3	3	3
Frame 1			2	2	2	2	2	2	2	2	2	4	4
Frame 2				3	4	4	4	5	5	5	5	5	5
		F	F	F	F			F			F	F	

Page Replacement Algorithm: History Based Algorithms

- The locality of references imply that recent past history of page access can allow the guessing of next page access
  - Guess which pages are least likely to be used in the near future
  - Select these pages for replacement
- Two history-based algorithms
  - Least-recently-used (LRU)
  - Least-frequently-used (LFU)
- The price must be paid
  - Not learning the history can be costly
  - Learning the history itself is also costly

Page Replacement Algorithm: LRU

- Selects the page with the longest time since the last reference
  - The page that has not been used for the longest period of time
  - This strategy has been adopted in many systems
- Disadvantage
  - Implementation requires substantial hardware to keep track of the usage history of the pages
  - Implementation methods include using a stack a time-of-use field, reference bit, additional-reference-bit, and second chance algorithm
    - Using a bit or two bits more in the page table as a simplification
  - Read the textbook for more details

Page Replacement Algorithm: LFU

- A counter keeps count of the number of references that have been made to each page
  - Addition column in the page table
  - Should keep the most active pages in memory

Example: Page Replacement Algorithm

Example: Page Fault Rate of Different Page Replace Algorithms 1

Consider the following page access sequence (reference string), evaluate the number of page faults of OPT, LRU and LFU if there are three frames available.

1 2 3 4 1 2 5 1 2 3 4 5

Example: Page Replacement Algorithm

Reference String **1 2 3 4 1 2 5 1 2 3 4 5**

The reference string describes the order of PAGEID requested by a process

OPT Page Replacement

Physical Memory

		1	2	3	4	1	2	5	1	2	3	4	5
Frame 0		1	1	1	1	1	1	1	1	1	3	3	3
Frame 1			2	2	2	2	2	2	2	2	2	4	4
Frame 2				3	4	4	4	5	5	5	5	5	5
		F	*F*	*F*	*F*			*F*			*F*	*F*	

Example: Page Replacement Algorithm

Physical Memory

LRU Page Replacement

		1	2	3	4	1	2	5	1	2	3	4	5
Frame 0		1	1	1	4	4	4	5	5	5	3	3	3
Frame 1			2	2	2	1	1	1	1	1	1	4	4
Frame 2				3	3	3	2	2	2	2	2	2	5
		F	F	F	F	F	F	F			F	F	F

Physical Memory

LFU Page Replacement

		1	2	3	4	1	2	5	1	2	3	4	5
Frame 0		1	1	1	4	4	4	5	5	5	3	4	5
Frame 1			2	2	2	1	1	1	1	1	1	1	1
Frame 2				3	3	3	2	2	2	2	2	2	2
		F	F	F	F	F	F	F			F	F	F

All three pages have been accessed one time. Any one of them may be replaced

Example: Page Replacement Algorithm

Reference String **1 2 3 4 1 2 5 1 2 3 4 5**

This reference string does not show locality of references

Example: Page Fault Rate of Different Page Replace Algorithms 2

Consider the following page access sequence (reference string), evaluate the number of page faults of LRU and LFU if there are three frames available.

1 1 2 0 0 4 3 0 1 2 5 1

Example: Page Replacement Algorithm

LRU Page Replacement

Physical Memory

		1	1	2	0	0	4	3	0	1	2	5	1
Frame 0		1	1	1	1	1	4	4	4	1	1	1	1
Frame 1			2	2	2	2	2	3	3	3	2	2	2
Frame 2					0	0	0	0	0	0	0	5	5
		F	F		F		F	F		F	F	F	

LFU Page Replacement

Physical Memory

		1	1	2	0	0	4	3	0	1	2	5	1
Frame 0		1	1	1	1	1	1	1	1	1	1	1	1
Frame 1			2	2	2	2	4	3	3	3	2	5	5
Frame 2					0	0	0	0	0	0	0	0	0
		F	F		F		F	F			F	F	

Implementation of Page Replacement Algorithms

- The algorithms based on history are often the better performers.
  - Do not suffer from Belady's anomaly
  - Belong to a class of algorithms known as stack algorithms.
  - Stack algorithms are algorithms that the set of pages in memory for N frames is always a subset (included the same set) of the page set that would be in memory with N+1 frames

Implementation of Page Replacement Algorithms

- Stack based Implementation
  - Page identifiers are put onto a stack, and recently referred page is moved to the top of the stack
  - The bottom of the stack would be the LRU page

Approximated Implementation of LRU Algorithm

- Approximated algorithms are developed which are based on the following.
  - Reference Bit
  - Additional Reference Bit
  - Second-Chance Algorithm
  - Enhanced Second-Chance Algorithm

Approximated Implementation of LRU Algorithm

- In enhanced second-chance algorithm, the modify bit is considered together with the reference bit
  - There are four different possibilities with the two bits

Reference Bit	***Modify Bit***	***Notes***
0	0	Best page to replace
0	1	Modified page, the page needed to be copied out before replacement
1	0	Recently used and likely to be used again
1	1	Recently used and modified.

performance issues of virtual memory

Performance Issues

- Virtual memory systems can result in really poor performance
  - The demand paging activity can be very high
  - High enough to cause CPU utilization to approach zero

Performance Issues

- The number of frames allocated to each process is the key to performance
  - Too many frames: use a lot of memory and reduce level of multi-programming
  - Too few frames: a lot of page faults

Performance Issues

- If the number of page frames allocated to a process falls below a certain threshold, page faults will quickly increase.
  - The memory manager will have to replace some pages that are in active use
  - These active pages will probably be needed again very soon
  - More page faults in order to bring these pages back

Performance Issues

- The faulting processes queue up for the paging disk, CPU utilization decreases
  - The CPU scheduler will allow more new processes to increase the degree of multiprogramming to compensate for the low CPU utilization
  - Even greater contention for physical memory and causes more page faults

Thrashing

- The system spends more time on paging than doing useful work
  - System throughput can reach zero
- Another cause of thrashing is through increasing the degree of multiprogramming, up to a stage where there are too many active processes (and therefore active pages).
  - Each new process is started with getting frames from active processes, and causing more page faults

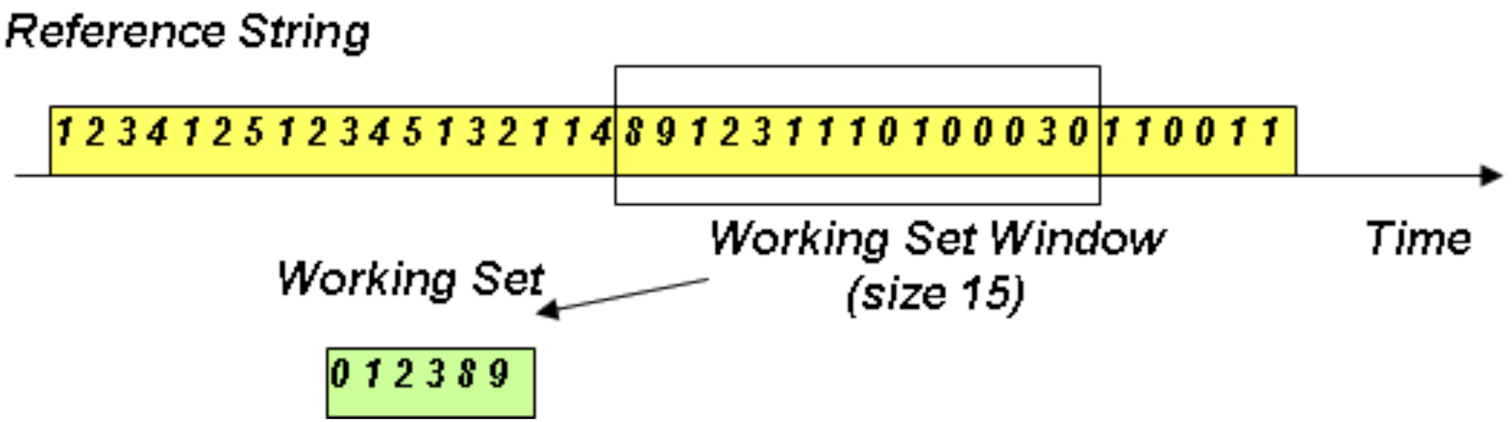
Solution to Thrashing

- A simple method to prevent thrashing is to provide enough frames for a process
  - The challenge is of course to work out how many frames are needed

Locality Principle and Working Set

- A working set window is the total number of page references within a period of time by a process
  - Locality principle means the page references should cluster
- The working set is defined as the set of page references contained in the most recent working set window
  - Suppose a process has the following page references during a particular period of time: 1116553331762122321235
  - With a working set window is 10, the working set is (1, 6, 5, 3)
  - Usual size of working set window is 10000

Locality Principle and Working Set



Dynamic Allocation of Frames

- An alternative method to prevent thrashing is based on page-fault frequency
  - Adjusting the number of frames allocated to a process based on the page-fault frequency of the process
  - A process with a high page-fault frequency would be allocated more frames because it needs more frames
  - A process with a low page-fault frequency would have a frame removed

Dynamic Allocation of Frames