Swap

What happens when memory runs out?

By Matthew Tancreti
For COM S 352
Iowa State University

Swap

Users have expectation they can run many programs concurrently

Memory requirements of all processes combined can easily exceed total physical memory

Observation

- At any time, many idle processes
- Only a few pages account for most memory accesses

Idea: rarely used pages can be stored on disk

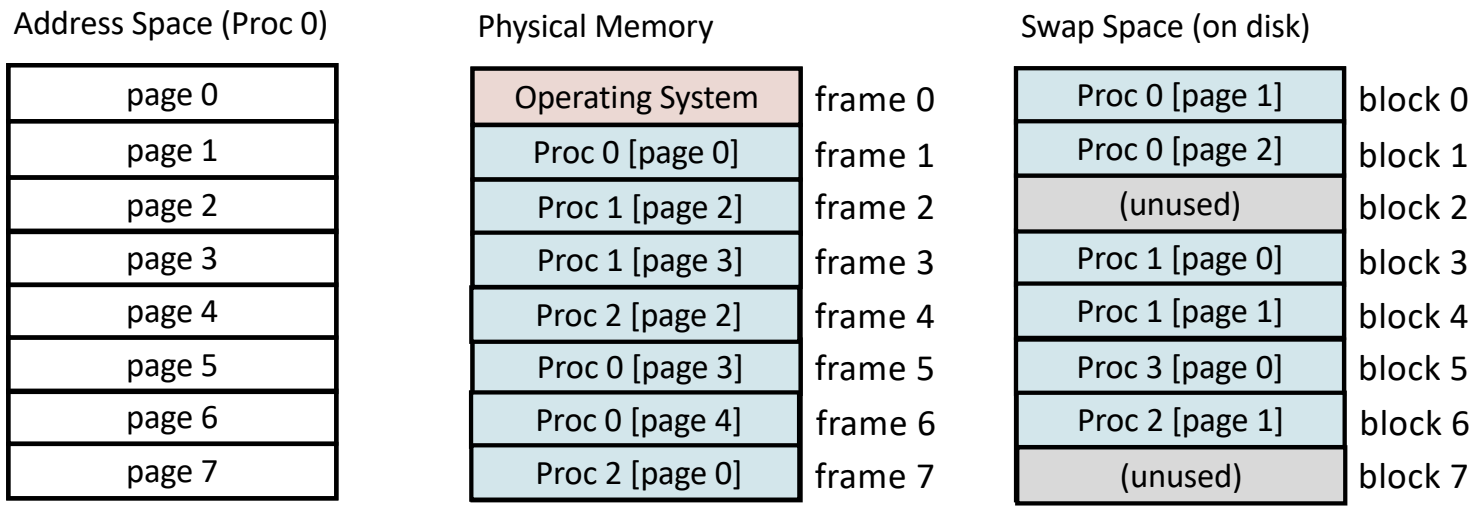
Rai stone [source]

How can OS take advantage of large slow disk space to give illusion of large address spaces?

Swap Space

When physical memory is full, less used pages moved to swap space

Swap space divided into **blocks** that can hold one page



Present Bit

Page table has **present bit** to indicate if page is in physical memory (1) or in swap (0)

Address Space (Proc 0)

page 0
page 1
page 2
page 3
page 4
page 5
page 6
page 7

Page Table (Proc 0)

VPN	PFN	valid	present
0	1	1	1
1	-	1	0
2	-	1	0
3	5	1	1
4	6	1	1
5	-	0	-
6	-	0	-
7	-	0	-

Physical Memory

Operating System	frame 0
Proc 0 [page 0]	frame 1
Proc 1 [page 2]	frame 2
Proc 1 [page 3]	frame 3
Proc 2 [page 2]	frame 4
Proc 0 [page 3]	frame 5
Proc 0 [page 4]	frame 6
Proc 2 [page 0]	frame 7

Swap Space (on disk)

Proc 0 [page 1]	block 0
Proc 0 [page 2]	block 1
(unused)	block 2
Proc 1 [page 0]	block 3
Proc 1 [page 1]	block 4
Proc 3 [page 0]	block 5
Proc 2 [page 1]	block 6
(unused)	block 7

What to do when page not present?

When program tries to access memory that is on a page currently in swap a **page fault** occurs

The page fault is just a trap, it is delt with by OS code called the **page-fault handler**

The page-fault handler needs to move the page back to main memory

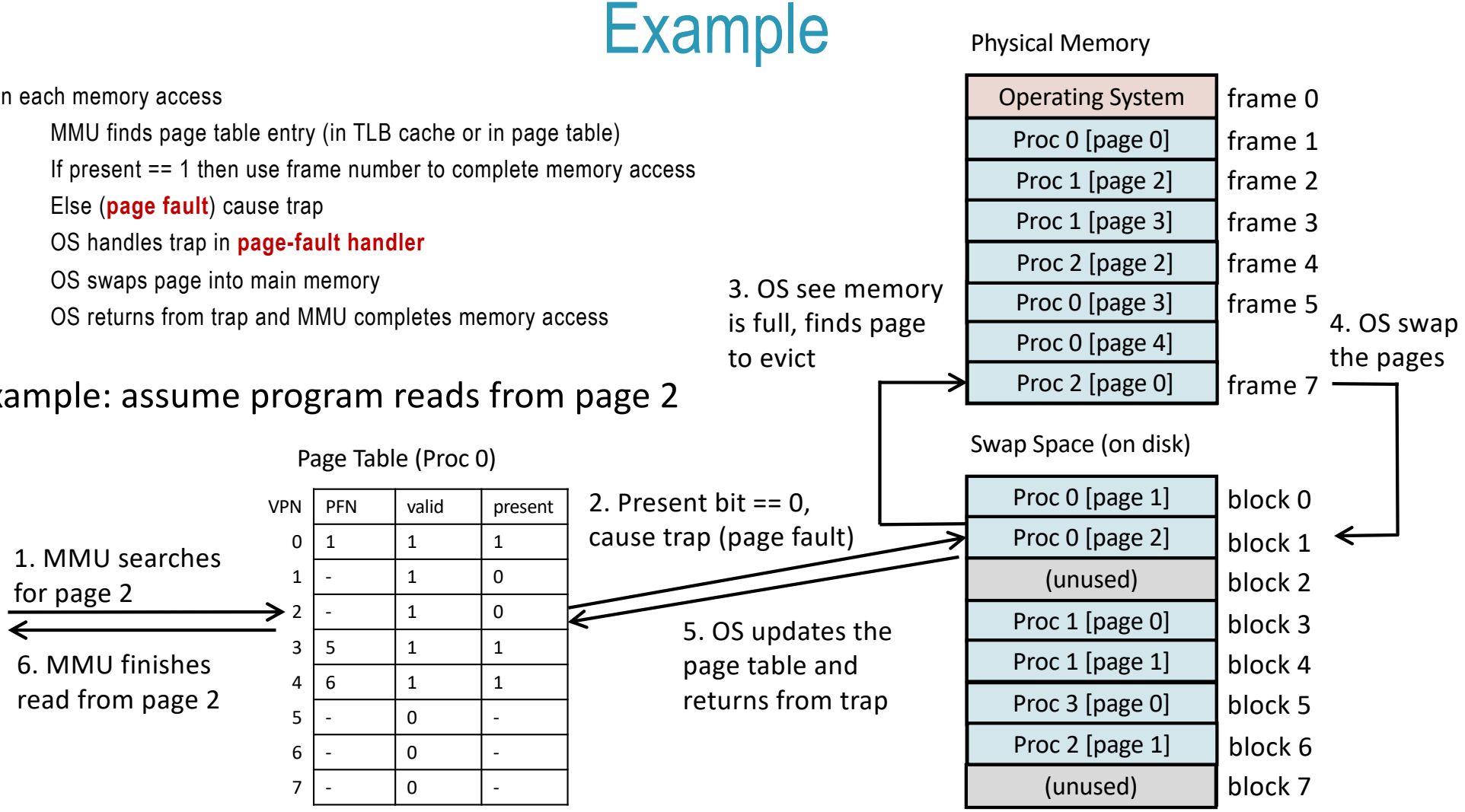
It may require making room by swapping some other page into swap space

Example

On each memory access

- MMU finds page table entry (in TLB cache or in page table)
- If present == 1 then use frame number to complete memory access
- Else (**page fault**) cause trap
- OS handles trap in **page-fault handler**
- OS swaps page into main memory
- OS returns from trap and MMU completes memory access

Example: assume program reads from page 2



Why Not Put Everything on Disk?

Main memory is 100x slower than CPU

SSD drive is 100x (or more) slower than main memory

Swap enables large easy to use address spaces, but cost of page fault is enormous, really want to avoid page faults

Important considerations

- How to decide which page to replace?
- What happens to performance when memory is full?

Replacement Policy

Need to decide on page to evict (**page out**) so a page can be brought back to memory (**page in**)

There are many **page-replacement policies**, just like we saw with TLB cache

- First In First Out (FIFO)
- Least Recently Used (LRU)
- Random
- ...

Proactive Free Space Management

Assume main main memory is full

- What happens when user start new application?
- Several page faults! Really slow!

OS proactively moves pages to swap to always keep a small amount for memory free for responding to sudden activity

Two thresholds **high watermark** and **low watermark**

- When free memory drops below low watermark OS start background task to start pushing pages to swap space
- Task stop when free memory passes high watermark