Solaris Memory Achitecgture

Kernel Hacking
CS 611

Demetrios Dimatos
Madhujith Hapuarachchi

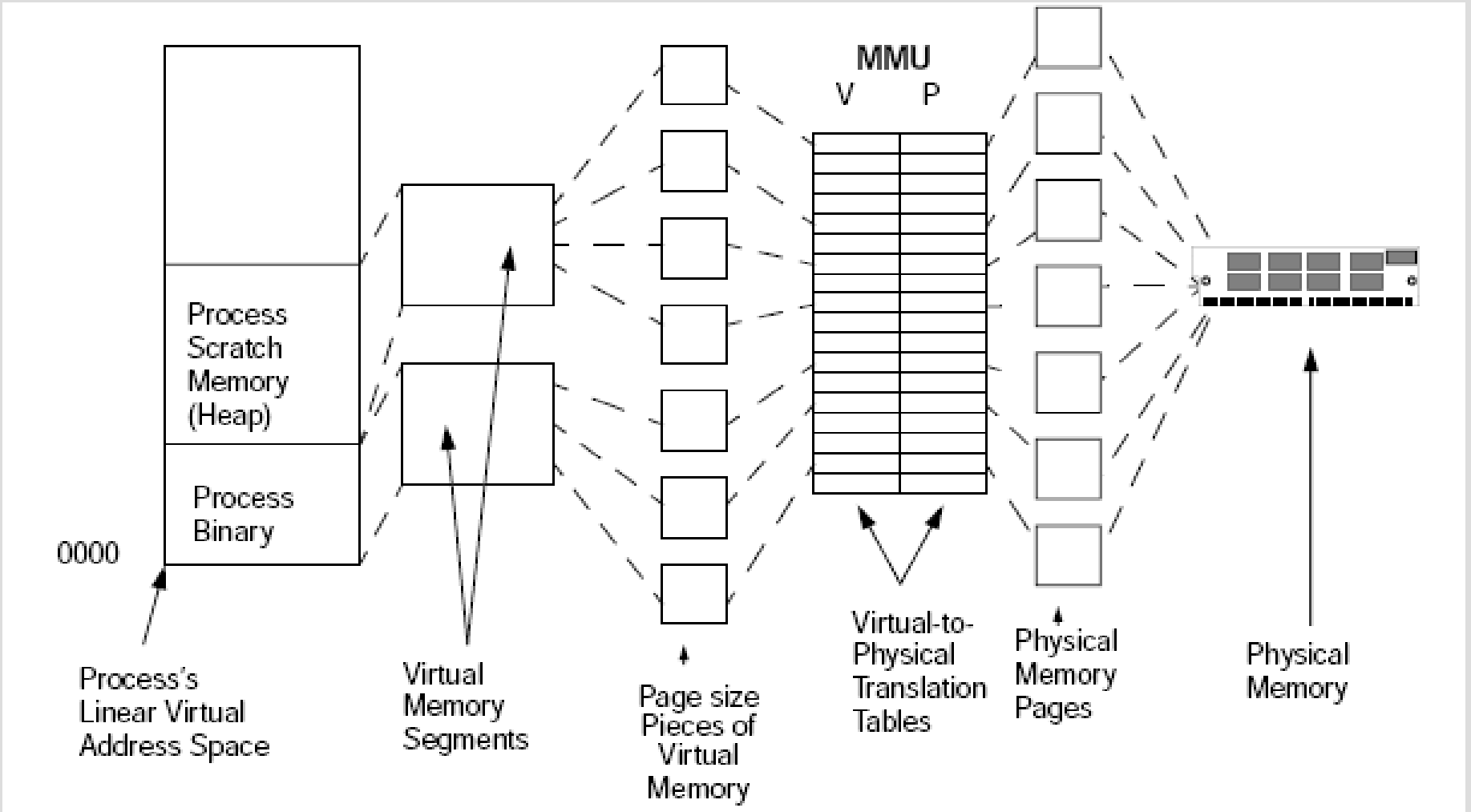
Overview

- Solaris Memory Management
- Types of Algorithms
- LRU Global Clock
- Scan Rate
- Paging Basics
- Memory Scheduler (Swapping)
- Questions

Solaris to Physical Memory

- Solaris breaks Virtual Memory into Segments
  - Example: binary segment, heap segment
  - Each segment manages mapping for the virtual address range and converts this mapping to MMU
- Hardware MMU maps these pages into physical pages buy using the platforms specific set of translation tables

Virtual Physical Memory Managment



Basic Types of Memory Management

- Swapping
  - Makes use of user process
    - All the pages of memory for the least active process will be freed
- Demand Paging
  - Uses a page for management
    - Swaps out small unused chunks, processes can still continue while less used parts can be swapped to quickly restore

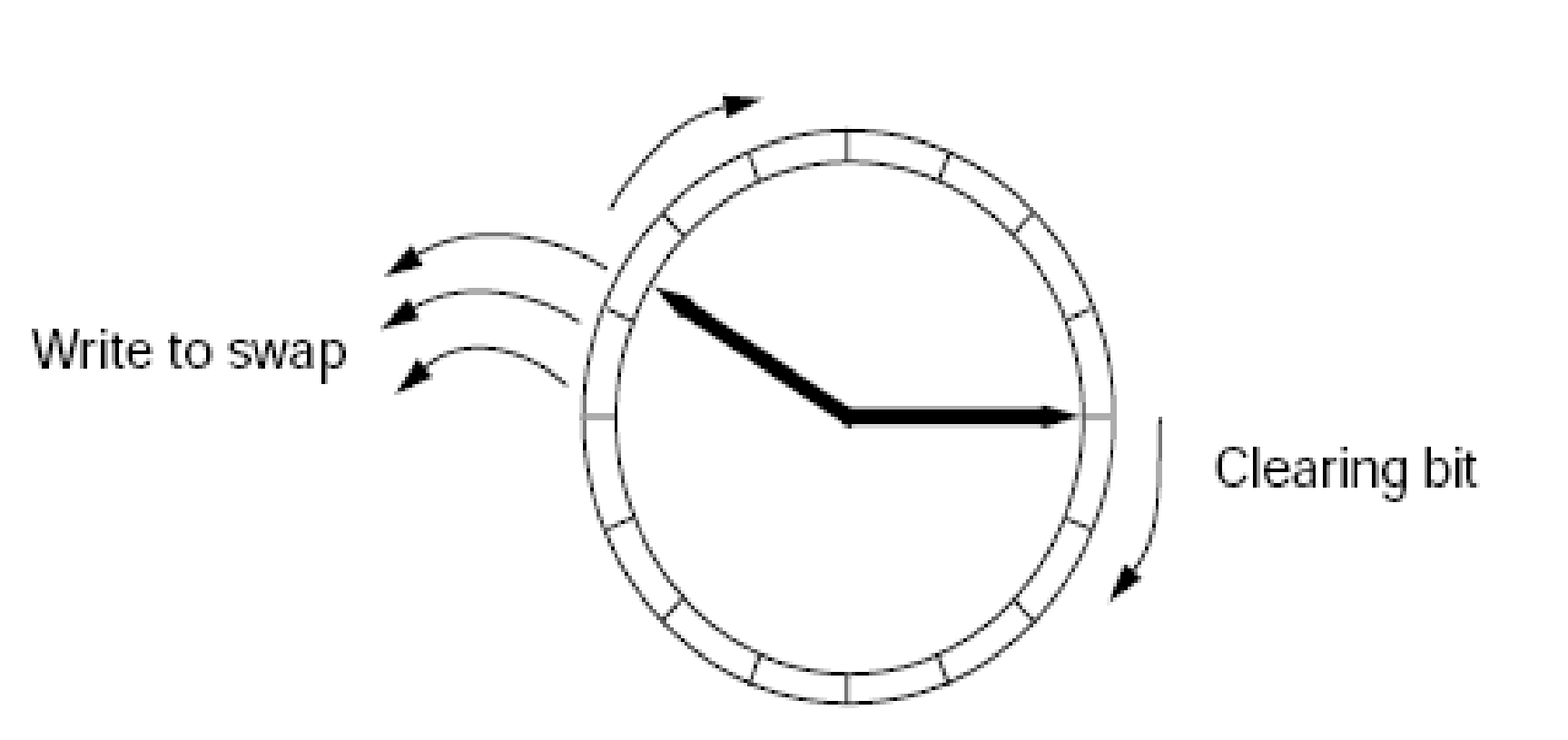
Virtual Memory Responsibility

- VM System implements
  - Responsible for I/O
  - Kernel
  - Applications
  - Shared Libraries
  - File Systems
- The VM system can allocate all free memory for I/O
  - Often reports that system is nearly ZERO memory, this is normal

Page Scanner

- Memory management daemon that manages system wide physical memory
- Scanner (pageout_scanner) tracks pages by reading the state of the hardware bits in MMU
  - MMU bits maintain state of referenced and written (dirty)
  - Uses a two-handed clock algorithm to determine eviction

Two-handed Clock Algorithm



Page-out Algorithm Parameters

- Distance between clock hands controlled by handspreadpages in terms of pages
  - Max setting for handpsreadpages is 8,192 pages (64 MB) and with systems with less than 128 MB memory it will be set to half of the memory

```
static pgcnt_t     handspreadpages = 0;
```

  - Note :- Solaris hardware page size is 8k.

Scan Rate Parameters

- Page scanner component is a kernel thread that is awakened when the amount of memory falls below ***lotsfree*** + ***deficit***.
- ***lotsfree*** is set at system start up, 1/64 of the systems memory
- ***deficit*** is usually set to zero or a small number of pages set by page allocator in anticipation of memory requests

Scan Rate

- The scan rate changes during the course of the page scanners operation
  - Scanners scans at ***slowscan*** when memory is below ***lotsfree*** and then increases to ***fastscan*** when memory has fallen below ***minfree*** threshold
  - ***slowscan*** is set to 100 pages by default
  - ***fastscan*** is set to ***physicalmemory/2*** capped at 8192 pages
- By converting memory to pages, a formula can be used to determine scanrate

Scan-rate Formula

$$scanrate = \left[\frac{(lotsfree - freememory)}{lotsfree} \right] + \left[slowscan * \left(\frac{freemem}{lotsfree} \right) \right]$$

$$scanrate = \left[\frac{(2048 - 1536)}{8192} \right] + \left[100 * \left(\frac{1536}{2048} \right) \right] = 2123$$

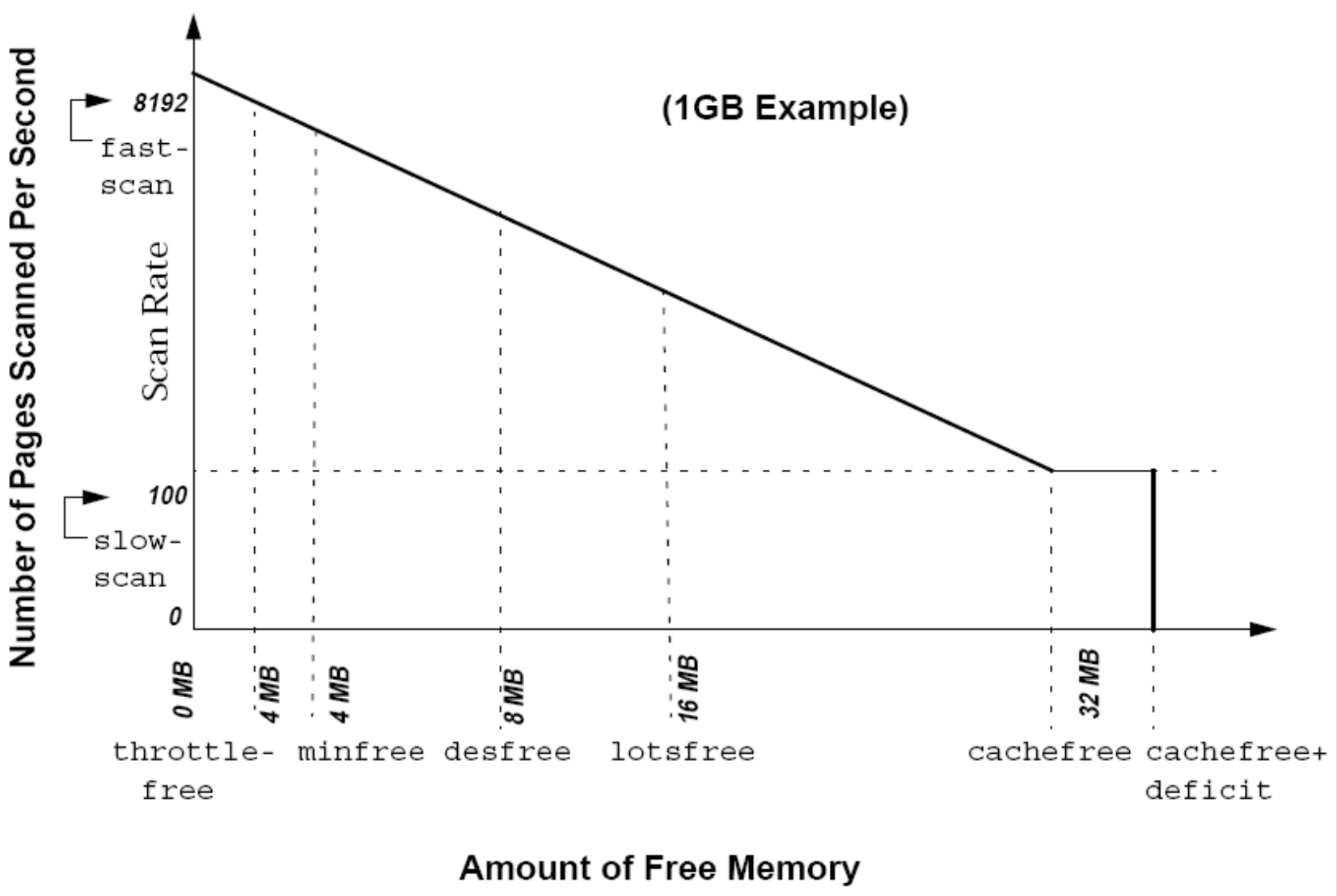
This yields a scan rate of 2123 pages per second

Scanner

- By default the scanner will run four times per second when there is a memory shortage
- If the amount of memory falls below ***cachefree***, scanner will start at 4 times a second
- If the amount of memory falls below ***desfree***, scanner will run at every clock cycle (100 times a second)
- This will keep at least ***desfree*** pages free on the free list

```
desfree = lotsfree / 2;
```

Scan Rate Interpolation



Priority Scanning Algorithm

- With high scan rates and constant requirement of free memory due to mass IO done by the system, scanner could take out any page that is free. Ex. Application pages, binaries, shared libraries, heap, stack (ones needed for swap)

- Improvement :- setting priorities to pages depending of current memory status making them eligible of un-swappable.

Priority Scanning Algorithm

- The new algorithm introduces a new paging parameter, ***cachefree***. When free memory lies between ***cachefree*** and ***lotsfree***,
  - **page scanner steals only file system cache pages.**
  - **scan rate algorithm is changed to the following**

$$scanrate = \left(\frac{cachefree - freememory}{cachefree} \times fastscan\right) + \left(slowscan \times \frac{freemem}{cachefree}\right)$$

CPU Utilization Clamp

- Overrides the scan rate
- prevents the page-out daemon from using too much processor time.

- ***min_percent_cpu*** – min available processor time
  - when free memory is at ***lotsfree*** (***cachefree*** with priority paging enabled)
  - Default – 4 %

- ***max_percent_cpu*** – max available processor time
  - if free memory were to fall to zero.
  - Default – 80 %

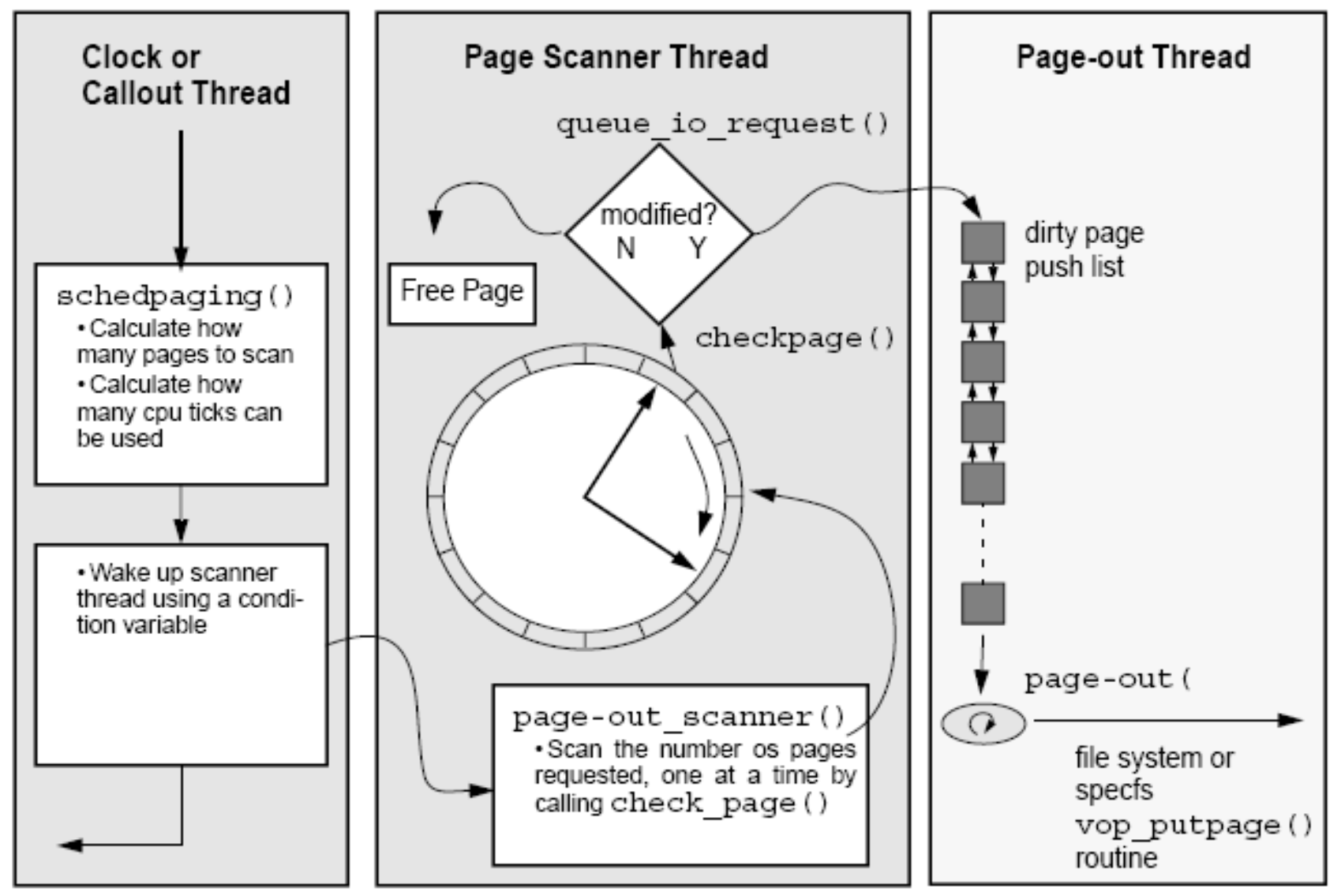
How Paging Works?

- The page scanner is implemented as two kernel threads, One thread scans pages, and the other thread pushes the dirty pages queued for I/O to the swap device

- Scanner is called :-
  - the kernel callout mechanism wakes the page scanner thread when memory is insufficient.
  - triggered by the **clock**() thread if memory falls below ***minfree***
  - called by the page allocator if memory falls below ***throttlefree***.

- The **schedpaging**() function checks whether free memory is below the threshold (***lotsfree*** or ***cachefree***) and, if required, prepares to trigger the scanner thread.

The Big Picture



Memory Scheduler (Swapping)

- CPU scheduler/dispatcher can swap out entire processes to conserve memory.
  - removing all of a process's thread structures and private pages from memory
  - setting flags in the process table to indicate that this process has been swapped out
- looks for processes that can be completely swap out when memory is consistently > ***desfree*** (30 sec ave)
- Two states :- **soft** swap and **hard** swap

Slow Swapping State

- Soft Swapping
  - Activated *when* 30-second average for free memory is below ***desfree***
  - Finds a process that has been sleeping for **maxslp**
  - Once found swaps out the thread structures for each
  - thread, then pages out all of the private pages of memory for that process.

Hard Swapping State

- Hard Swapping
  - At least two processes are on the run queue, waiting for CPU.
  - The average free memory over 30 seconds is consistently less than ***desfree***.
  - Excessive paging (determined to be true if page-out + page-in > ***maxpgio***) is going on.

- The kernel is requested to unload all modules and cache memory that are not currently active, then processes are sequentially swapped out until the desired amount of free memory is returned