Linux Kernel Programming
Memory Management Pt.2

Sang-Hoon Kim

Systems Software Research Group @ Virginia Tech

April 13, 2017

Everyday questions

- I found my smartphone/laptop is low on free memory
- This app greatly increases the free memory from my phone!! Five stars!!
- I hate Google Chrome because it hogs memory
- Linux installer suggests a swap partition which is twice of the RAM size
  - By the way, I have 32GB RAM and 128GB SSD... what?
- I don't understand the performance evaluation results
  - 300 MB/sec from hard disk drive????
- I could `malloc()` 2 GB from 1 GB Raspberry Pi...?

Myth

- High memory utilization makes me feel bad ☹
  - Swapping will destroy the system performance
  - Will increase the memory allocation time
  - Will trigger *some* background operations

- Are they all true?
- Will be happy if 20% is used and the rest is free?
- What do you want to do with the *"free memory"*?

Outline

1. Metrics for memory utilization
2. Page cache
3. Page reclamation

Outline

1. Metrics for memory utilization
2. Page cache
3. Page reclamation

System memory utilization

Metrics for memory utilization

- The portion of memory that is currently used in the system
- Memory in use + free memory = total memory
- Allocated memory / total memory or
  (total memory - free memory) / total memory
- E.g., using 3 GB out of 4 GB: 3 GB / 4 GB = 75%

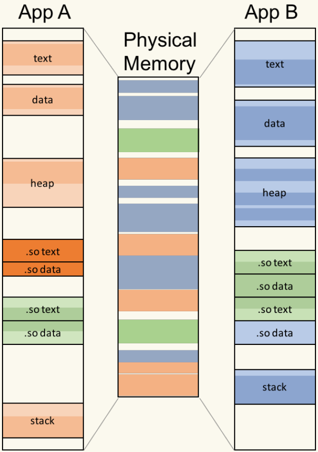
Per-process memory utilization

Metrics for memory utilization

- Processes request the kernel for virtual memory areas (VMAs)
- A read causes the corresponding page to be mapped to the zero page.
- A write causes the copy-on-write to the zero page
  - Allocate the actual page
  - Update the mapping to the new page
- VMAs can be shared between processes
- The sharing can be private

Per-process memory utilization

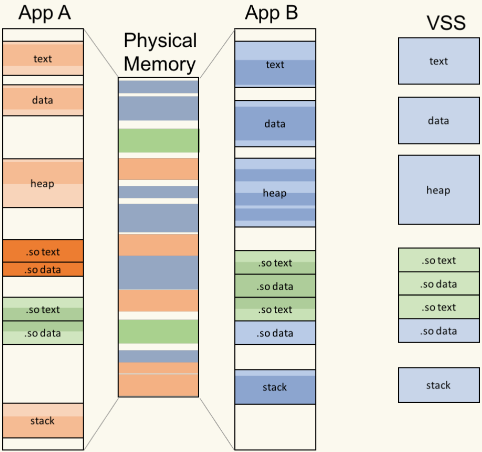
Metrics for memory utilization



Virtual set size (VSS)

Metrics for memory utilization

- The size of virtually allocated memory



Virtual set size (VSS)

Metrics for memory utilization

- The size of virtually allocated memory
- Usually, way to large than the actual memory footprint
- Can obtain by summing the size of each VMA
  - $\sum_{\text{Each VMA}} |\text{VMA}|$

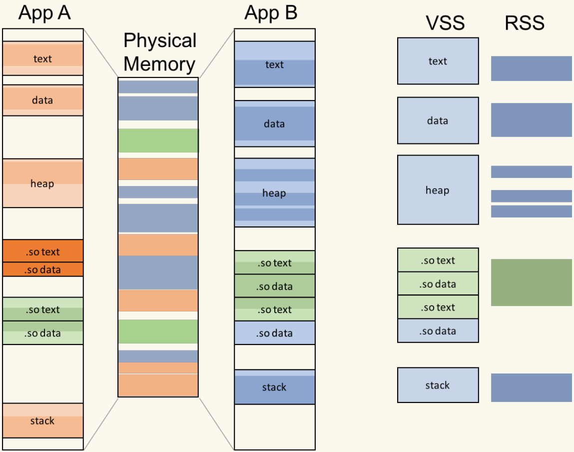
```
/* in include/linux/mm_types.h */

struct mm_struct {
  /* ... */
  unsigned long total_vm; /* Total pages mapped */
  /* ... */
};
```

Resident set size (RSS)

Metrics for memory utilization

- The memory footprint that the process actually occupies



Resident set size (RSS)

Metrics for memory utilization

- The memory footprint that the process actually occupies
- Can get by counting the valid page table entries
  - Page size × # of mapped pages

```
/* In include/linux/mm.h */

static inline unsigned long get_mm_rss(struct mm_struct *mm);
```

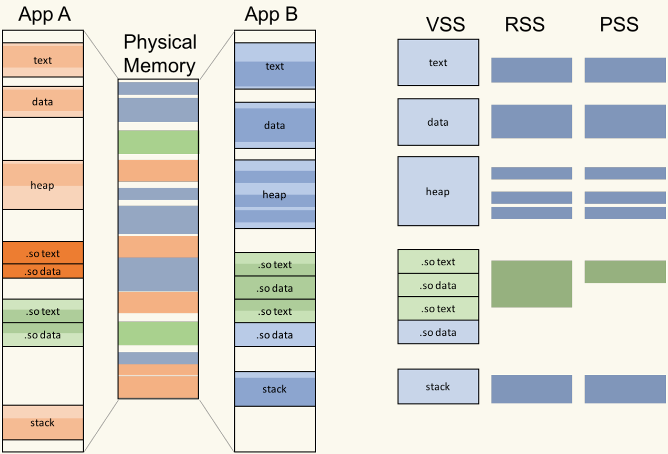
Proportional set size (PSS) / Unique set size (USS)

Metrics for memory utilization

- VSS and RSS do not consider the memory sharing between processes
- Note: VMA is the unit of memory sharing
- Proportional set size (PSS)
  - Account for the number of sharing processes
  - Indicate the memory contribution of the process
  - Page size $\times \sum_{\text{Each VMA}}$ # of mapped pages / # of share
  - E.g., VMA contains 10 pages and shared by 4 processes
    = 4 KB $\times$ 10 / 4 = 10 KB
- Unique set size (USS)
  - The memory footprint that is not shared with any other processes
  - The amount of memory that can be reclaimed by killing the process

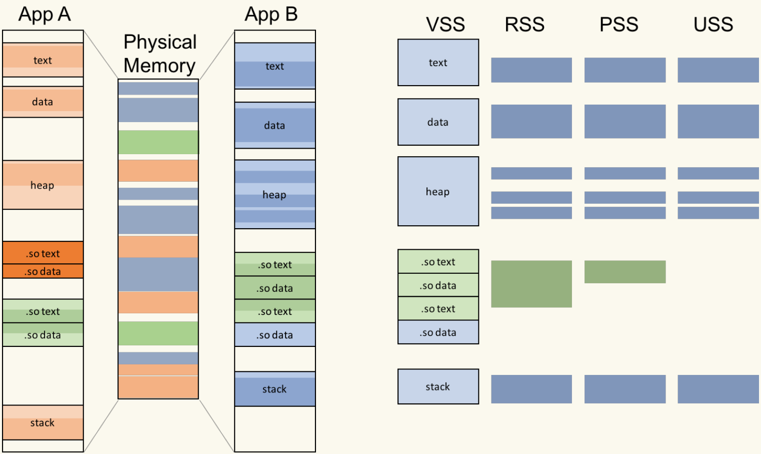
Proportional set size (PSS) / Unique set size (USS)

Metrics for memory utilization



Proportional set size (PSS) / Unique set size (USS)

Metrics for memory utilization



Accessing the metrics

Metrics for memory utilization

- `top`

```
3. sanghoon@cerberus: ~
sanghoon@cerberus: ~
top - 12:04:53 up 58 days, 23:10,  2 users,  load average: 0.00, 0.02, 0.00
Tasks: 292 total,   1 running, 291 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 12300996 total,  3241476 free,   153492 used,  8906028 buff/cache
KiB Swap: 15616892 total, 15567456 free,    49436 used. 11428304 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3854 sanghoon  20   0   42064   3896   3008 R   0.7  0.0   0:00.60 top
 3796 sanghoon  20   0   45380   4828   3884 S   0.0  0.0   0:00.04 systemd
 3799 sanghoon  20   0   63412   1972      0 S   0.0  0.0   0:00.00 (sd-pam)
 3841 sanghoon  20   0   99504   3108   2148 S   0.0  0.0   0:00.01 sshd
 3842 sanghoon  20   0   22832   5336   3208 S   0.0  0.0   0:00.11 bash
```

Accessing the metrics

Metrics for memory utilization

- /proc/[pid]/smap

```
00400000-004f2000 r-xp 00000000 08:01 21233800          /bin/bash
Size:                    968 kB
Rss:                     888 kB
Pss:                     177 kB   /* 888 / 177 = 5 */
Shared_Clean:            888 kB   /* Rss == Shared */
Shared_Dirty:              0 kB

...

006f1000-006f2000 r--p 000f1000 08:01 21233800          /bin/bash
Size:                      4 kB
Rss:                       4 kB
Pss:                       4 kB

...

7f92acb43000-7f92acd42000 ---p 0000b000 08:01 14682101    /lib/x86_64-linux
     -gnu/libnss_files-2.19.so
Size:                   2044 kB
Rss:                       0 kB
Pss:                       0 kB
```

Outline

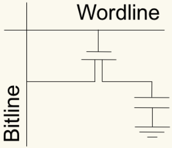
1 Metrics for memory utilization

2 Page cache

3 Page reclamation

Is high memory utilization bad?
Page cache

- In fact, you are wasting your memory if you don't utilize it
- Keep dissipating 5–30% of system energy [1, 2, 3]
  - The cells in DRAM are effectively capacitors
  - The value is determined by the amount of electron charges (i.e., voltage) in the cell
  - Should be refreshed periodically ($\sim$64ms)



- Free pages and used pages are all the same from the memory modules' perspective
- Miss the opportunity to utilize the *fastest* storage in the system that you/OS can fully control its contents
  - Explicitly utilize as RAM disks
  - Else?

Page cache

Page cache

- Utilize free pages to cache the data from slow storage devices
- Implemented at the virtual file system (VFS) layer
  - Will be discussed in the following lecture
- Maintain file-backed pages
  - Pages are indexed by ({device, file}, offset)

- When `read()/fread()` from a file,
  - Look up the page cache
  - If exist, copy the data from the page cache page. Fast!!
  - If not,
    - Allocate page(s)
    - Put the page into the page cache and lock properly
    - Generate I/O request(s) to corresponding block device/file system
    - Copy the read content to the user-space buffer

Page cache

Page cache

- When `write()/fwrite()` to a file,
  - Look up the page cache
  - If exist, apply the write to the page and flag it dirty. Fast!!
  - If not,
    - Allocate page(s)
    - Generate I/O request(s) to corresponding block/file system
    - Apply the write to those pages, make them dirty
  - Dirty pages will be *magically* written back to the storage
  - The write-back makes the page clean

Page cache

Page cache

- Page cache significantly improves the system performance
  - Can read without accessing the slow storage devices
  - Can buffer writes
  - Merge overwrites before writing to the storage
  - Can shape the write pattern to the storage device
- In enterprise and HPC environments, performance-critical servers are equipped with a huge amount of memory for the page cache

How large is the page cache?

Page cache

```
sanghoon@echo_debian:~$ free -m
              total       used       free     shared    buffers     cached
 Mem:         32133      26789       5344         24       1592      23720
 -/+ buffers/cache:       1475      30658
 Swap:            0          0          0
```

- Memory utilizatoin = 26,789 / 32,133 = 83%
- Page cache = 23,720 MB (88% / 74%)
- Available = 5,344 + (1,592 + 23,720) $\pm\epsilon$ = 30,656 MB

```
sanghoon@cerberus_ubuntu:~$ free -m
              total       used       free     shared  buff/cache   available
Mem:          12012        156       3158         88        8697       11154
Swap:         15250         48       1520
sanghoon@echo:~$
```

- Memory utilizatoin = (156 + 8,697) / 12,012 = 74%
- Page cache = +-8,697 MB (98% / 72%)
- Available = 3,158 + 8,697 $\pm$ ϵ = 11,154 MB

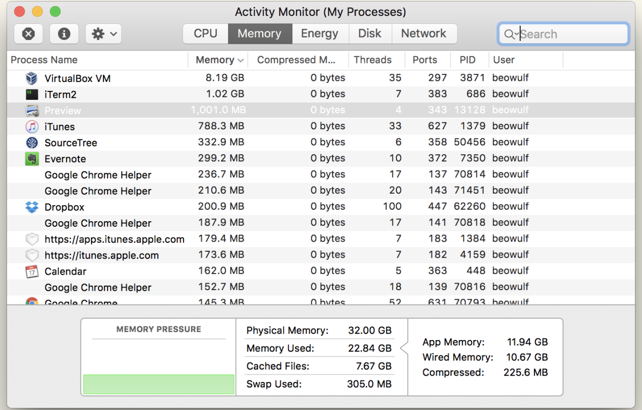
How large is the page cache?

Page cache

```
# cat /proc/meminfo
MemTotal:       32904980 kB
MemFree:         4971852 kB
MemAvailable:   32399416 kB
Buffers:         1634068 kB
Cached:         24768688 kB <--
  ...
```

How large is the page cache?

Page cache



Activity Monitor (My Processes)

CPU | Memory | Energy | Disk | Network

Process Name	Memory	Compressed M...	Threads	Ports	PID	User
VirtualBox VM	8.19 GB	0 bytes	35	297	3871	beowulf
iTerm2	1.02 GB	0 bytes	7	383	686	beowulf
Preview	1,001.0 MB	0 bytes	4	343	13128	beowulf
iTunes	788.3 MB	0 bytes	33	627	1379	beowulf
SourceTree	332.9 MB	0 bytes	6	358	50456	beowulf
Evernote	299.2 MB	0 bytes	10	372	7350	beowulf
Google Chrome Helper	236.7 MB	0 bytes	17	137	70814	beowulf
Google Chrome Helper	210.6 MB	0 bytes	20	143	71451	beowulf
Dropbox	200.9 MB	0 bytes	100	447	62260	beowulf
Google Chrome Helper	187.9 MB	0 bytes	17	141	70818	beowulf
https://apps.itunes.apple.com	179.4 MB	0 bytes	7	183	1384	beowulf
https://itunes.apple.com	173.6 MB	0 bytes	7	182	4159	beowulf
Calendar	162.0 MB	0 bytes	5	363	448	beowulf
Google Chrome Helper	152.7 MB	0 bytes	18	139	70816	beowulf
Google Chrome	145.3 MB	0 bytes	52	631	70793	beowulf

MEMORY PRESSURE

Physical Memory:	32.00 GB
Memory Used:	22.84 GB
Cached Files:	7.67 GB
Swap Used:	305.0 MB

App Memory:	11.94 GB
Wired Memory:	10.67 GB
Compressed:	225.6 MB

QnA

Page cache

Q: I don't understand the performance evaluation results

- 300 MB/sec from hard disk drive????

A: Check whether the data is accessed from the page cache

- Unfair between the 1st run and subsequent runs
- Drop or warm up the page cache
- You can explicitly drop page caches
- Bypass the page cache by `open()` with `O_DIRECT`

```
/* To free pagecache: */
$ echo 1 > /proc/sys/vm/drop_caches

/* To free reclaimable slab objects (includes dentries and inodes): */
$ echo 2 > /proc/sys/vm/drop_caches

/* To free slab objects and pagecache: */
$ echo 3 > /proc/sys/vm/drop_caches
```

QnA

Page cache

Q: I found my smartphone/laptop is low on free memory

A: Don't panic

- Your operating system is doing its best to maximize the system performance
- By the way, check the swap status for sure

Q: This app increases the free memory from my phone!! Five stars!!

A: ...

Outline

1. Metrics for memory utilization
2. Page cache
3. Page reclamation

Memory is limited

Page reclamation

- Eventually, memory will be filled up with page cache pages and process-allocated pages
- The kernel cannot keep allocating pages
- Should reclaim allocated memory somehow

- When the kernel should start the reclamation?
- Who will reclaim the memory?
- Which memory should be reclaimed?
- How reclaim?

When to start the memory reclamation?

Page reclamation

- When fail to allocate a page from `alloc_pages()`
  - Mostly due to zone imbalancing (e.g., alloc_page(GFP_DMA))
  - `ZONE_DMA, ZONE_DMA32, ZONE_NORMAL, ZONE_HIGHMEM`
- Based on the amount of free memory

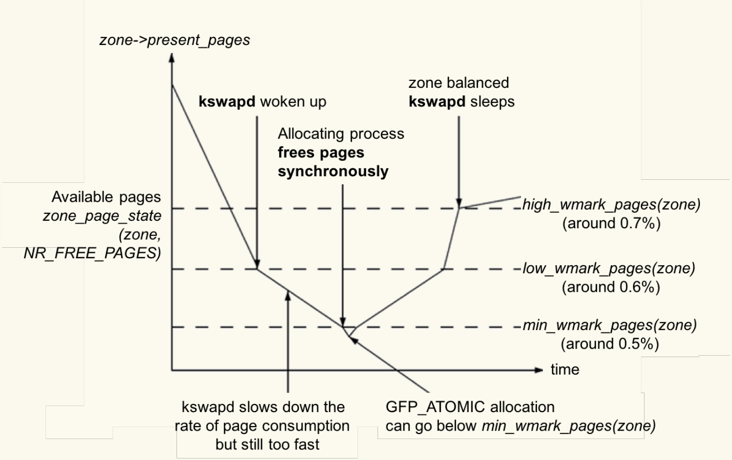
Who will reclaim?

Page reclamation

- Direct reclamation
  - When `alloc_pages()` fails
  - Synchronously reclaim pages for critical page allocation
- Background reclamation
  - Triggered when `alloc_pages()` fails
  - A kernel thread a.k.a `kswapd` reclaims the page frames
  - Keep the free memory above thresholds

Background reclamation

Page reclamation



What to reclaim?

Page reclamation

- Should evict file pages
  - Memory is for anonymous pages originally
  - Reclaiming clean file pages is fast
    - Drop the page from the page cache
  - Require slow I/Os to swap-out and -in anonymous pages
  - Many files are only accessed once
    - E.g., multimedia files
- Should keep file pages
  - Many clean pages are performance critical (e.g., code)
  - The write-back might block the reclamation
  - Cold anonymous pages are really cold
  - Storage devices are (were) extremely slow

What to reclaim?

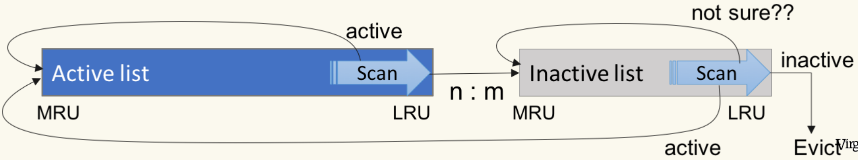
Page reclamation

- Keep *important* pages
  - Keep important pages whatever they are for
  - Can apply the traditional cache management policies
    - LRU, LFU, LRFU, CLOCK, ARC, CAR, MQ, ...
  - Anonymous pages and file-back pages have different characteristics
    - Many files are only accessed once
    - Code pages are performance critical
    - Cold anonymous pages are really cold
- OK. let manage them separately

2Q

Page reclamation

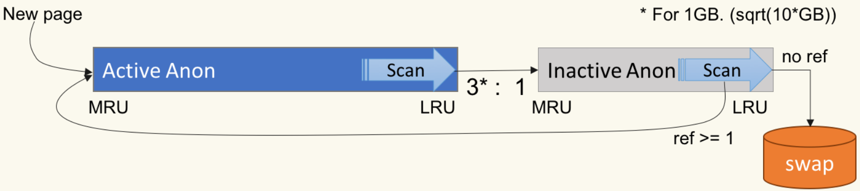
- None of OS/cache uses the LRU/LFU in practice
  - Updating the position upon every access requires a significant overhead
- Linux employs a modified 2Q policy
  - Known good to separate inactive pages from active ones
  - Maintain an LRU list for inactive pages and an LRU list for active pages
  - Scan a part of the lists
  - Migrate pages between the lists according to some policies



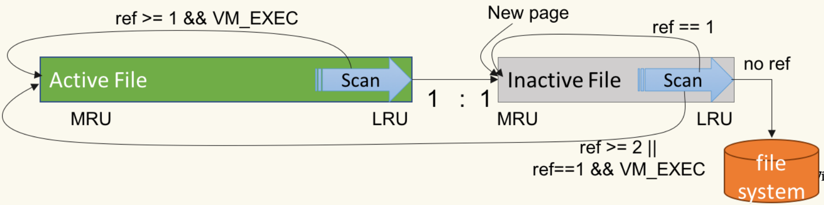
Managing the page lists

Page reclamation

- Anonymous pages



- File pages



Swappiness

Page reclamation

- Tendency to reclaim anonymous pages with swap-out
- Integer between 0 to 200 (60 by default)
  - 60 means anon:file = 60:140
  - 100 means anon:file = 100:100
- Effectively 0 when no swap device exists
- Can control via `/proc/sys/vm/swappiness`

Have a look at the source code

Page reclamation

- `mm/vmstat.c` and `mm/vmscan.c`
- Isolate pages from the lists
  - `shrink_inactive_list()`
  - `shrink_active_list()`
- Process the isolated page
  - `shrink_page_list()`

Check the size of page lists

Page reclamation

```
# cat /proc/meminfo
MemTotal:        32904980 kB
MemFree:          4971852 kB
MemAvailable:    32399416 kB
  ...
Active:          14175700 kB
Inactive:        12260264 kB
Active(anon):       34020 kB
Inactive(anon):     24728 kB
Active(file):    14141680 kB
Inactive(file): 12235536 kB
  ...
```

- Comment: We cannot tell whether the pages in the active/inactive page list are actually active/inactive or not
  - The activeness can be determined by examining the page
  - The pages evicted from the inactive list are inactive.

How to reclaim?

Page reclamation

- File pages
  - Clean: Drop from the page cache
    - The data is in the filesystem
  - Dirty: May drop after writing them back, or keep it
- Anonymous pages
  - Perform the swap-out

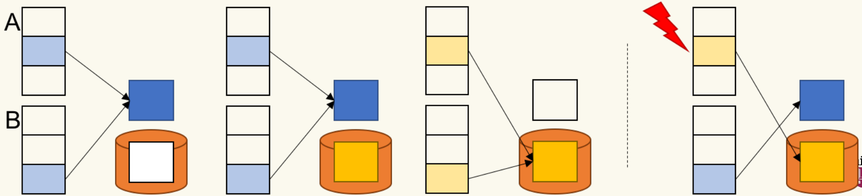
Swap-out v0.1

Page reclamation

- Note: A page can be shared between processes

- Allocate a swap entry from the swap device
- Write the page to the swap entry
- For each PTE points to the page (Q: How?)
  - Mark the PTE as invalid and update to the swap entry

- What if a process accesses the swapped-out page during the swap-out?

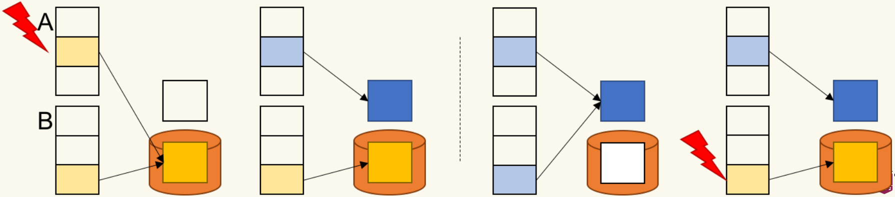


Swap-in v0.1

Page reclamation

- When a process accesses the swapped out PTE
- Allocate a page
- Read in the page from the swap device
- Update the PTE accordingly

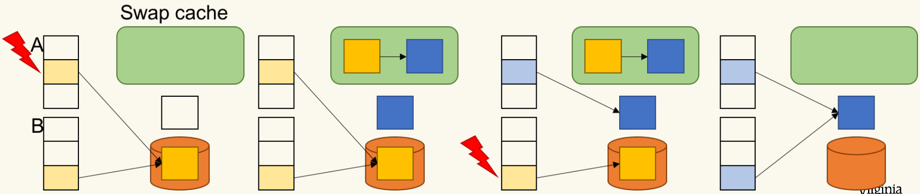
- Should the kernel fix the PTE from other processes and release swap entry?
- Which one is the current one?



Swap cache

Page reclamation

- Prevent the race conditions during the swap-in and out
- Keep (swap entry ID, page) mappings for transient pages
- Consult to the swap cache first to swap-in
- If a corresponding mapping exists, update the PTE to the page in the memory instead of loading the swap entry



Conclusion

- Do not under-utilize your memory
- Trust the techniques that are reviewed, evaluated, and proved

- Yes, memory management is dirty and complicated
- However, it is connected to everything
- Non-volatile memory and flash memory provide new opportunities

Bibliography I

[1] Carroll, A., and Heiser, G.
An analysis of power consumption in a smartphone.
In *Proceedings of the 2010 USENIX Annual Technical Conference (USENIX ATC'10)* (2010), pp. 21–35.

[2] Carroll, A., and Heiser, G.
The systems hacker's guide to the galaxy: Energy usage in a modern smartphone.
In *Proceedings of the 4th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'13)* (2013).

[3] Liu, S., Pattabiraman, K., Moscibroda, T., and Zorn, B. G.
Flikker: Saving DRAM refresh-power through critical data partitioning.
In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVI)* (2011), pp. 213–224.