CS 477
Advanced Operating System

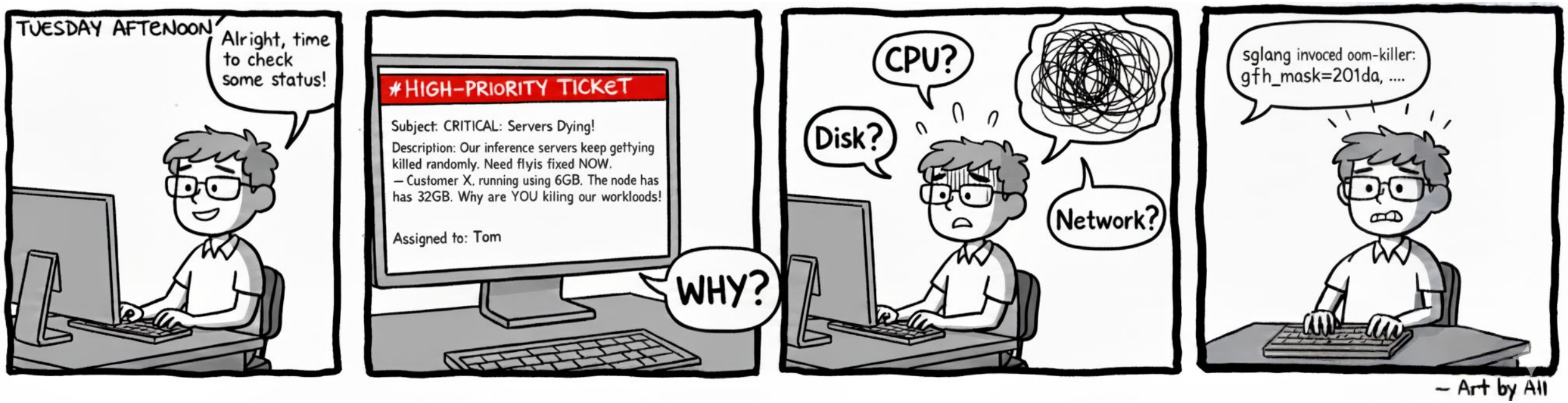
Tutorial 10: Handling memory pressure on Linux

Today's tutorial

- Memory pressure
  - Why should you care about / learn a bit about it?
  - What tool can you use to investigate?
  - What will happen under memory pressure?
    - When will reclamation happen?
    - What is gonna happen in reclamation?
    - What memory is going to be reclaimed?
- Finer - grained resource control beyond system-level
  - Cgroup v2 is your friend
  - Monitoring metrics
  - Brief explanation of containers
- Demo code: https://github.com/PanJason/memory_demo

Why shall I learn these things, like complex mm?

You just start as a cloud platform engineer. It is another Tuesday. You just settled in when a high priority ticket lands in your inbox.



If you understand mm, you will start to check:

1. Kernel OOM, cgroup OOM?
2. Is reclamation working correctly?
3. Then, maybe cgroup config problem?

Start with a simple program allocating memory

Start with a very simple example

- Keep allocating 1GB of pages (anonymous)

Demo code:
https://github.com/PanJason/module_and_ebpf/tree/main/kernel_module_examples/hello_world

Where can I check the impact of this program?
$cat /proc/meminfo
$cat /proc/vmstat # ultimate source
$cat /proc/pressure/memory
$sar -B 1

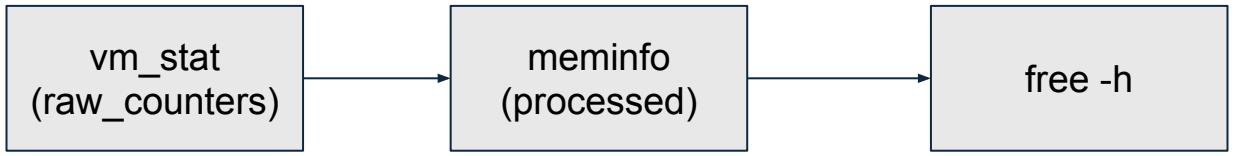
How to understand these metrics?

```
MemTotal:       1056306592 kB
MemFree:         728859540 kB
MemAvailable:   1033404508 kB
Buffers:            2998324 kB
Cached:           300242480 kB
SwapCached:             0 kB
Active:           113563816 kB
Inactive:         192330040 kB
Active(anon):       2662368 kB
Inactive(anon):       78176 kB
Active(file):     110901448 kB
Inactive(file):   192251864 kB
Unevictable:         27576 kB
Mlocked:             27576 kB
SwapTotal:        8388604 kB
SwapFree:         8388604 kB
Zswap:                  0 kB
Zswapped:               0 kB
Dirty:                400 kB
Writeback:             44 kB
AnonPages:        2680920 kB
Mapped:            555928 kB
Shmem:              78868 kB
KReclaimable:     7558704 kB
Slab:             9555440 kB
SReclaimable:     7558704 kB
SUnreclaim:       1996736 kB
KernelStack:        57216 kB
PageTables:         47464 kB
SecPageTables:          0 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:    532445900 kB
Committed_AS:   128037380 kB
VmallocTotal:   1374389534719
VmallocUsed:        1611908 kB
VmallocChunk:           0 kB
Percpu:            575488 kB
HardwareCorrupted:      0 kB
AnonHugePages:          0 kB
ShmemHugePages:         0 kB
ShmemPmdMapped:         0 kB
FileHugePages:          0 kB
FilePmdMapped:          0 kB
Unaccepted:             0 kB
HugePages_Total:     4000
HugePages_Free:      4000
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        2048 kB
Hugetlb:          8192000 kB
DirectMap4k:      4965736 kB
DirectMap2M:     99432448 kB
DirectMap1G:    969932800 kB
```

G1

G2

G3



G4

Kernel Doc:
https://www.man7.org/linux/man-pages/man5/proc_meminfo.5.html

G5

To better read these metrics, I did a “manual” classification

G1: Total mem info

G2: Page cache + Swap cache info

G3: LRU info

G4: Swap info

G5: Slab info

G6

G6: Huge page info

Another good monitoring tool

Sar: which also reads data from /proc/meminfo

$ sar -B 1

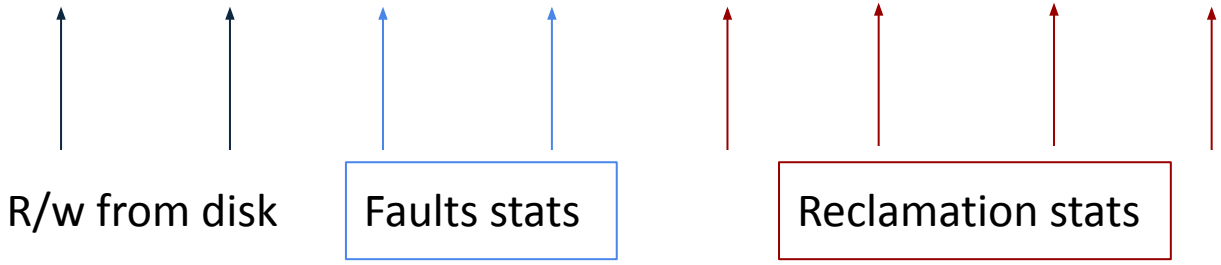
$ sar -b 1

https://man7.org/linux/man-pages/man1/sar.1.html

```
Linux 6.5.0-14-generic (rs3labsrv8)     11/09/2025      _x86_64_        (256 CPU)

06:38:16 PM  pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s    %vmeff
06:38:17 PM      4.00      0.00  11499.00      0.00  29152.00      0.00      0.00      0.00      0.00
06:38:18 PM      0.00      0.00   8945.00      0.00   5899.00      0.00      0.00      0.00      0.00
06:38:19 PM      0.00     20.00   5838.00      0.00   6045.00      0.00      0.00      0.00      0.00
06:38:20 PM     40.00    456.00  11642.00      0.00  17492.00      0.00      0.00      0.00      0.00
06:38:21 PM   2700.00   3052.00  11140.00      0.00  17300.00      0.00      0.00      0.00      0.00
06:38:22 PM   5008.00      4.00  47123.00      0.00  51590.00      0.00      0.00      0.00      0.00
^C

Average:      1292.00    588.67  16031.17      0.00  21246.33      0.00      0.00      0.00      0.00
```



The important thing in memory - reclamation

We saw for the previous example, reclamation happens.
Why do we care about reclamation?

- If memory is enough, everything works fine
- Otherwise it can impact the upper layer business metrics
  - Disk I/O can be slow
  - Allocation can be stalled
  - Workload can be killed

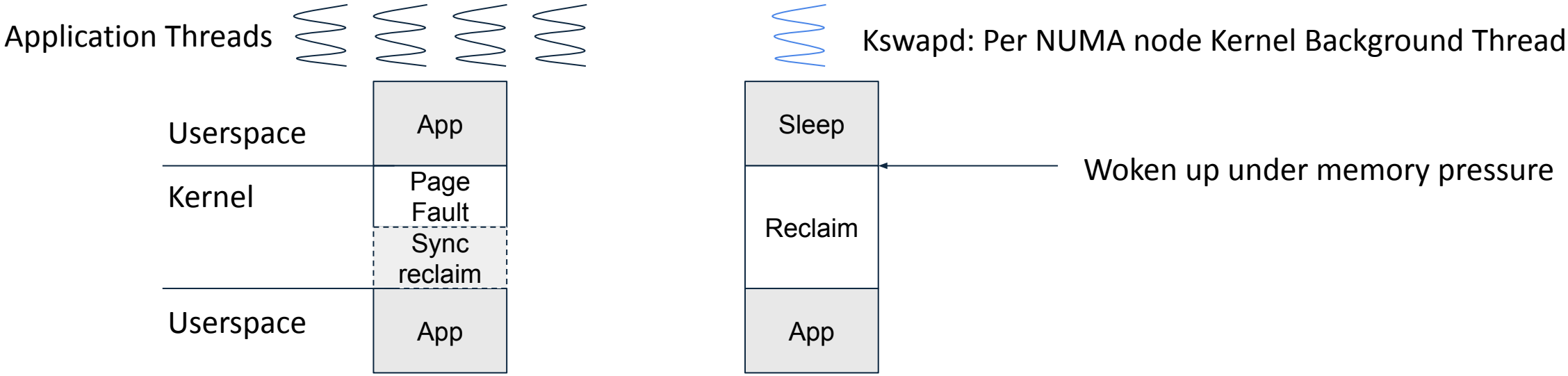
To understand reclamation, there are several questions:

1. When is the reclamation going to happen?
2. What is going to happen in reclamation?
3. What memory is going to be reclaimed?

When is the reclamation going to happen?

There are two types of reclamation in Linux kernel

1. Asynchronous reclamation: kswapd
   a. $ ps -o pid,comm,%mem,%cpu -p $(pgrep kswapd)
   b. Try to reclaim memory at the background
2. Synchronous reclamation: inside page fault
   a. If allocation is so fast that kswapd cannot keep up, page fault will trigger synchronous reclamation
3. Async is usually fine. Sync is problematic



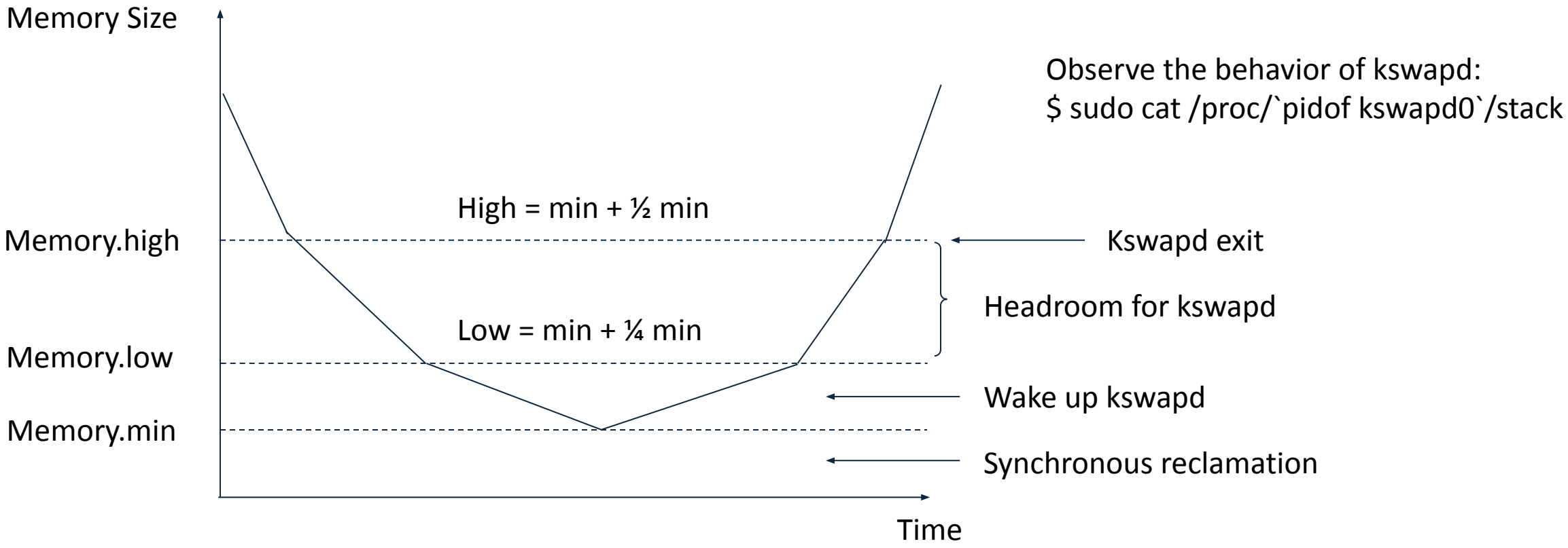
When is the reclamation going to happen?

System-level reclamation:

$ cat /proc/sys/vm/min_free_kbytes

$ echo xx | sudo tee /proc/sys/vm/min_free_kbytes # Distributed across zones

$ cat /proc/zoneinfo



Override the system-level behavior with Cgroup

Besides system-wise behavior, cgroup can override the config.
Cgroup serves as a very important mechanism to isolate resource
It allows user to explicitly control resources such as

- CPU
- Memory
- IO
- …

Check:
$ cat /sys/fs/cgroup/cgroup.controllers

Some useful documentation:
https://facebookmicrosites.github.io/cgroup2/docs/overview.html
https://docs.kernel.org/admin-guide/cgroup-v2.html

Cgroup v2 basic operations

Cgroup v2 enabling:

- Add systemd.unified_cgroup_hierarchy=1 cgroup_enable=memory cgroup_memory=1 to boot cmdline

Cgroup create:

- $ sudo mkdir /sys/fs/cgroup/demo

Cgroup set limit:

- echo 4G | sudo tee /sys/fs/cgroup/demo/memory.max

Cgroup related limit:

- memory.min => guaranteed memory size
- memory.low => won't be reclaimed with best effort
- memory.high => start sync reclaim, apply jiffy (async in non-task context)
- memory.max => start sync reclaim, can oom

What if reclaim makes no progress?

Out-of-memory (OOM) killing can happen in the following cases:

1. Direct reclamation is not allowed (GFP_ATOMIC)
2. Direct reclamation is allowed but cannot make enough process (All mem being used)

OOM is the last way to resolve memory pressure.

OOM picks up target based on badness calculation:

https://elixir.bootlin.com/linux/v6.17.7/source/mm/oom_kill.c

There are some control knobs:

$ cat /proc/[pid]/oom_score_adj

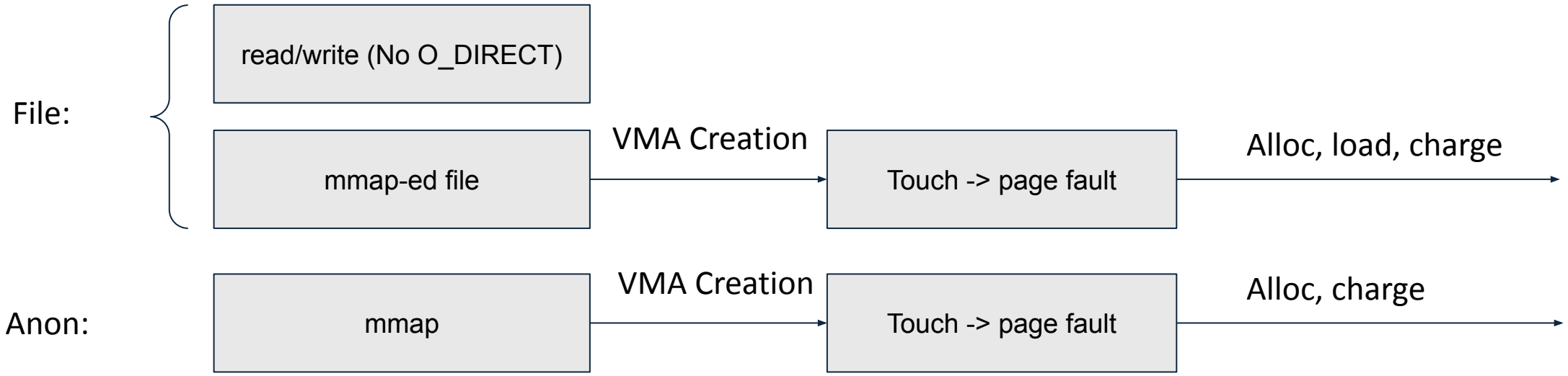
Where to check the oom info?

- Dmesg usually

What is going to happen in reclamation?

There are two types of pages to be reclaimed

- File pages (page cache)
- Anonymous memory (only when swap is enabled)

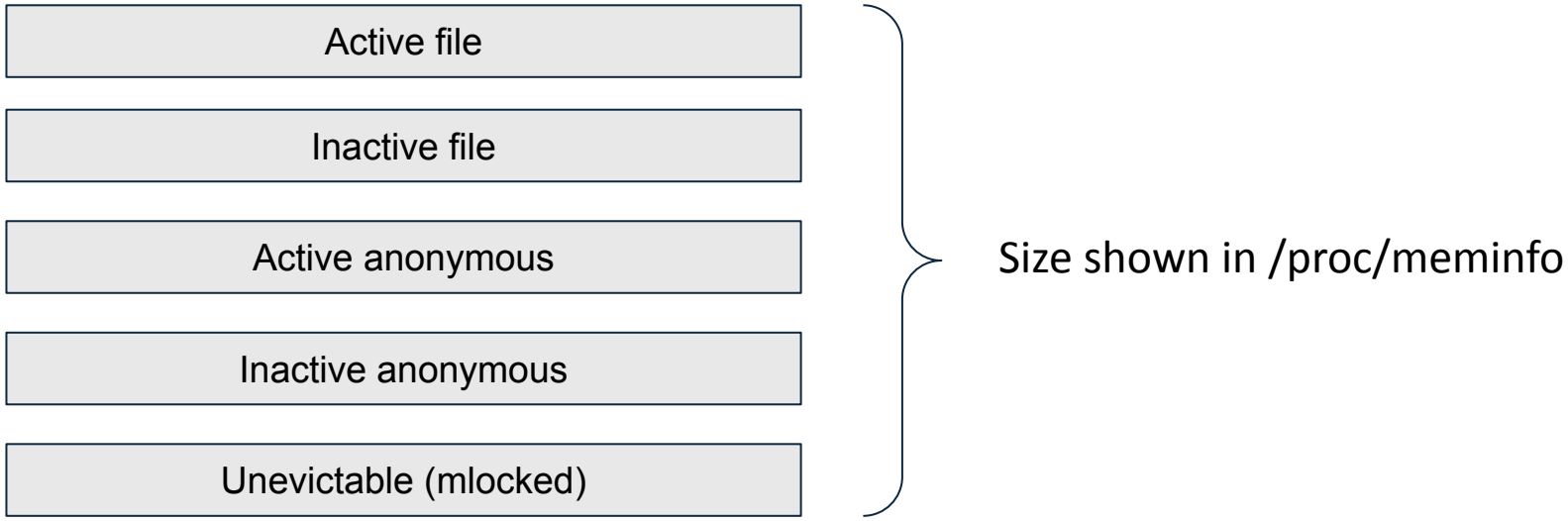


What is going to happen in reclamation?

There are two types of pages to be reclaimed

- File pages (page cache)
- Anonymous memory (only when swap is enabled)

Kernel organize these pages into 5 lists (before MGLRU) called lruvec



What is going to happen in reclamation?

Tuning knobs:

$ cat /proc/sys/vm/swappiness # Control the ratio between file and anon

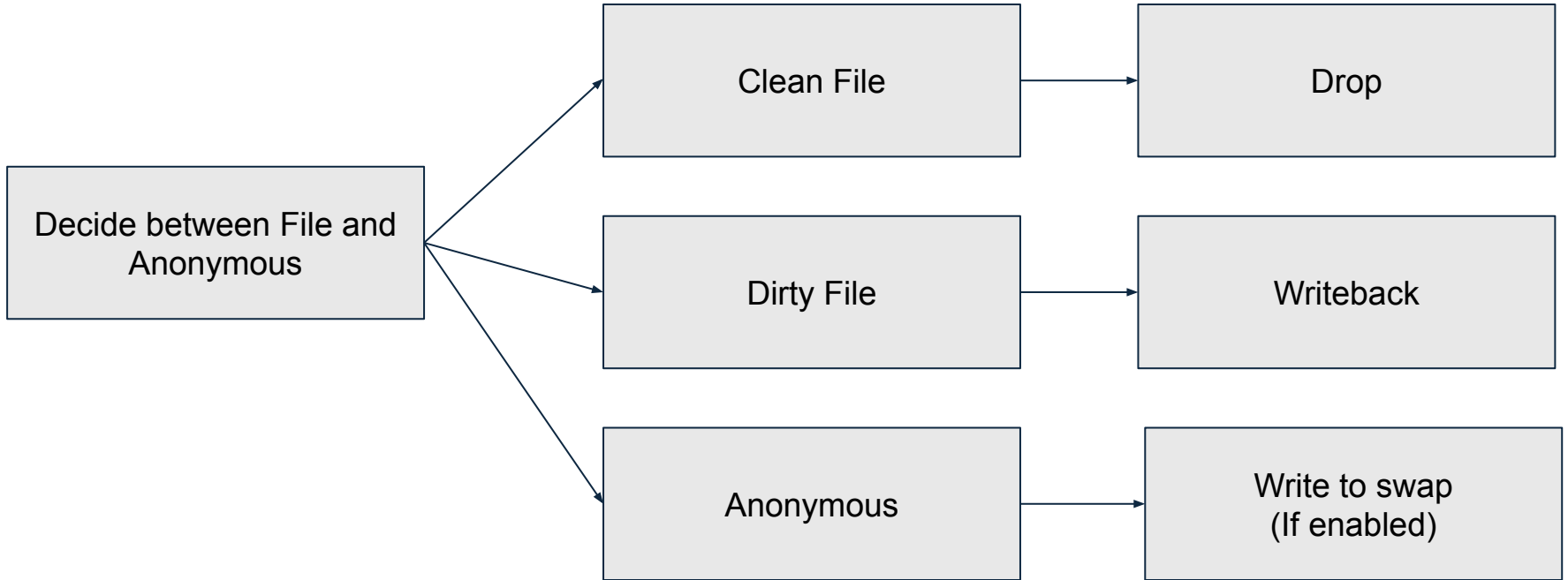
$ cat /proc/sys/vm/dirty_ratio # Above which block the process creating dirty

$ cat /proc/sys/vm/dirty_background_ratio # Start to writeback at the background

Monitoring loads:

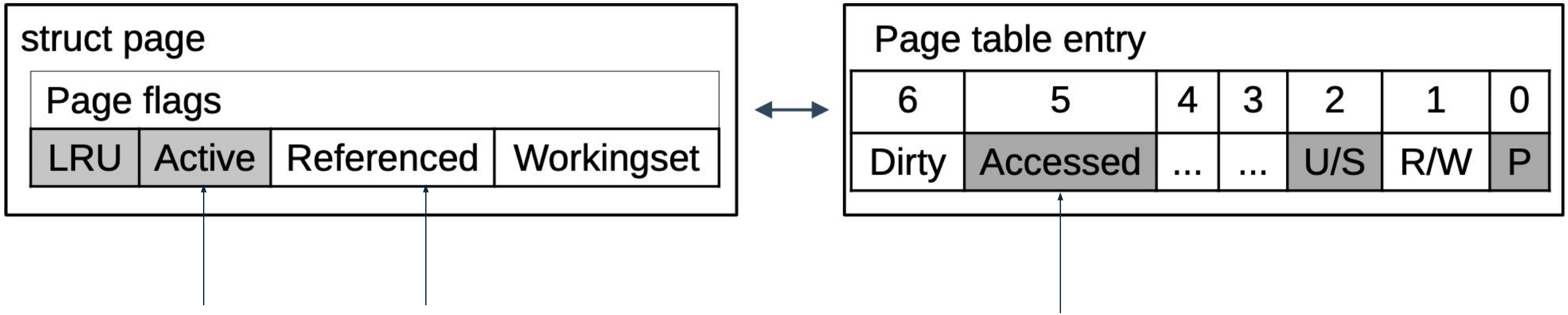
$ iostat -x 1

$ watch -n 1 'cat /proc/meminfo | grep -E "Dirty|Writeback"'



What memory is going to be reclaimed?

Reclamation function scans a given lruvec and decides for each page where it should go => recall the pgscan and pgsteal metrics
Some related PTE bits and page flags



If inactive

Set after clear accessed bit so we know in next scan this page was accessed previously

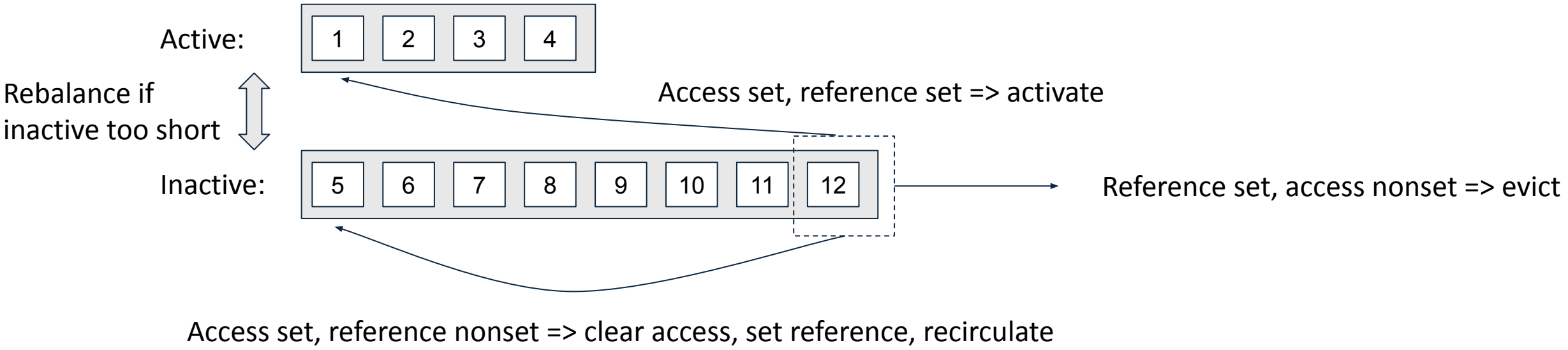
Set by the hardware when accessing a page

What memory is going to be reclaimed?

Key idea: protect hot workingset and filter out one-hit-wonder quickly

1. Newly fault-in => inactive
2. Accessed once => inactive
3. Accessed twice => active
4. Inactive too short => rebalance

In the real implementation => more complicated



What other things you can do with reclamation?

Save memory cost with transparent memory offloading!
Memory is costly in today's data center.
Swap is a way to save memory cost.
But swap is costly, so we have to bound the swapin cost.
Introduce the memory pressure metric

- Some
- Full

$ cat /proc/pressure/memory

Control some under 1% so the swap overhead to application is bounded