# Reading memcg Stats More Efficiently

… and possibly cpu & io stats

JP Kobryn
LPC 2025

# Context

Cgroup stats are collected across our fleet ***continuously***.

One major tool used for telemetry is “below”.

- https://github.com/facebookincubator/below

General CPU **Mem** I/O Pressure Properties Network

| **Name** | Mem | Mem Swap | Anon | File | Kernel | Kernel Stack | Slab | Sock |
|---|---|---|---|---|---|---|---|---|
| <root> | 236.7 GB | 324.1 MB | 13.7 GB | 222.4 GB | 5.7 GB | 205.9 MB | 4.9 GB | 24 K |
| └+ app.slice | 228 KB | 0.0 B | 0.0 B | 0.0 B | 228 KB | 0.0 B | 1.7 KB | 0.0 |
| └+ dev-hugepages.mount | 96 KB | 0.0 B | 0.0 B | 88 KB | 8 KB | 0.0 B | 4.5 KB | 0.0 |
| └+ dev-mqueue.mount | 4 KB | 0.0 B | 0.0 B | 0.0 B | 4 KB | 0.0 B | 1.2 KB | 0.0 |
| └+ hostcritical.slice | 17.1 GB | 84.5 MB | 153.7 MB | 16.7 GB | 226.7 MB | 3 MB | 215.4 MB | 0.0 |
| └+ init.scope | 75.2 MB | 4 KB | 12.1 MB | 37.8 MB | 25.2 MB | 1.2 MB | 23.6 MB | 0.0 |
| └+ overcommit.slice | 0.0 B | 0.0 B | 0.0 B | 0.0 B | 0.0 B | 0.0 B | 0.0 B | 0.0 |
| └+ proc-sys-fs-binfmt_misc.mount | 12 KB | 0.0 B | 0.0 B | 0.0 B | 12 KB | 0.0 B | 5.1 KB | 0.0 |
| └+ session.slice | 448 KB | 0.0 B | 0.0 B | 0.0 B | 448 KB | 0.0 B | 0.0 B | 0.0 |
| └+ sys-fs-fuse-connections.mount | 8 KB | 0.0 B | 0.0 B | 0.0 B | 8 KB | 0.0 B | 2.2 KB | 0.0 |
| └+ sys-kernel-config.mount | 4 KB | 0.0 B | 0.0 B | 0.0 B | 4 KB | 0.0 B | 1.8 KB | 0.0 |
| └+ sys-kernel-debug.mount | 4 KB | 0.0 B | 0.0 B | 0.0 B | 4 KB | 0.0 B | 1.2 KB | 0.0 |
| └+ sys-kernel-tracing.mount | 8 KB | 0.0 B | 0.0 B | 0.0 B | 8 KB | 0.0 B | 1.2 KB | 0.0 |
| └+ system.slice | 109.7 GB | 204.9 MB | 8 GB | 99.1 GB | 2.5 GB | 143.8 MB | 1.8 GB | 8 KB |

# cat /sys/fs/cgroup/cpu.stat

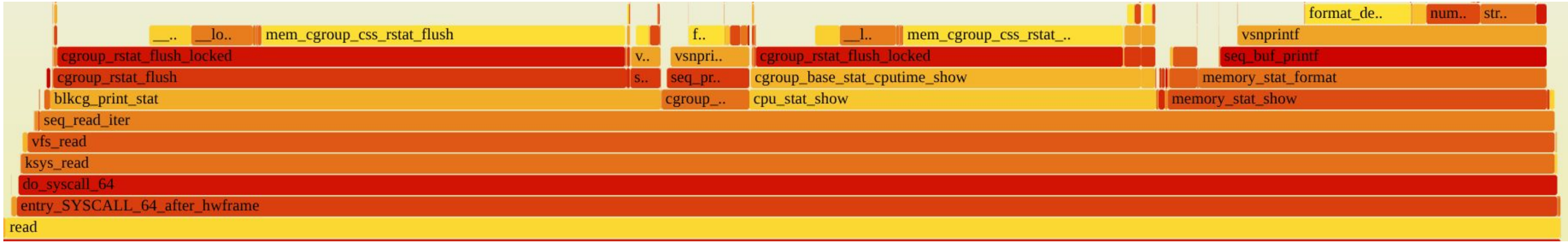
Any optimizations in reading cgroup stats when periodically sampled over a large enough fleet can lead to significant power savings.

Workstream:

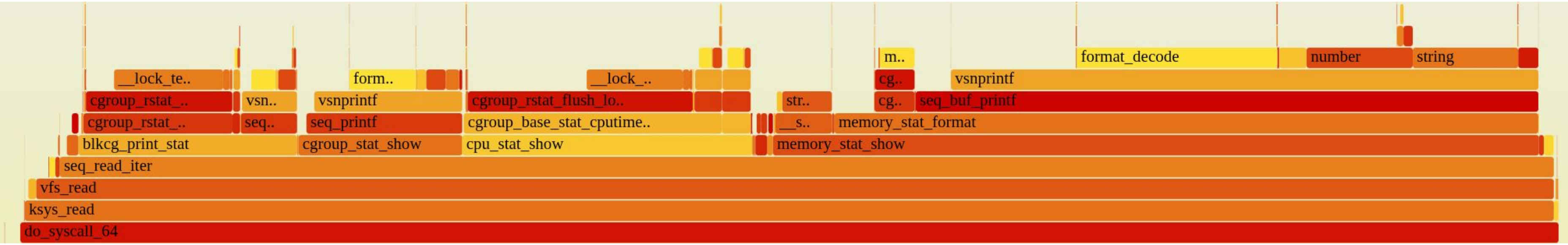
- Completed
  - Avoid indirect flushing of memory stats when flushing cpu and io stats.
- Pending upstream
  - Avoid the overhead of numeric/string conversions between kernel and userspace.
- In progress
  - Minimize reader overhead and latency of flushing.
    - Avoid readers paying the cost of flushing.
    - Avoid contention among parallel readers.

# Reading memory cgroup stats notoriously expensive

Pre 6.16, flushing cpu or io stats caused memory stats to be flushed:




Since 6.16, flushing of cpu, io, memory stats are now independent:



# Benchmark - Parallel Kernel Builds in Separate Cgroups

***control:***

real 1m55.079s

user 0m1.147s

sys 1m53.153s

***experiment:***

real 1m0.840s

user 0m0.999s

sys 0m59.413s

***control:***

34.52% mem_cgroup_css_rstat_flush

4.24% __blkcg_rstat_flush

0.07% cpu_stat_show

***experiment:***

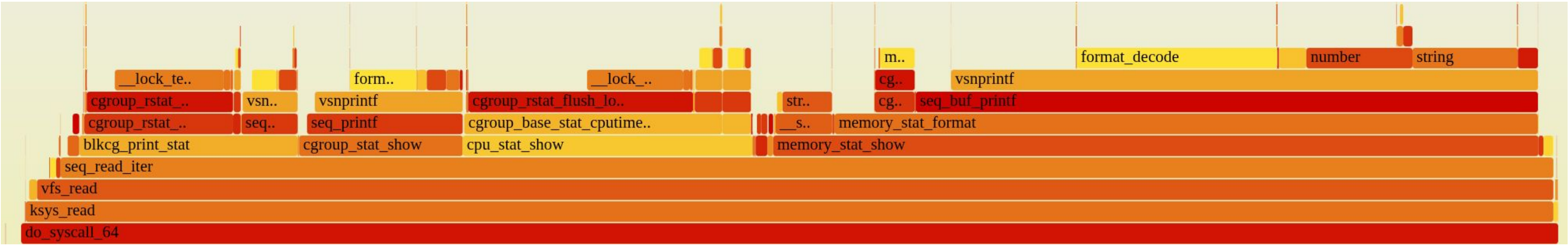
2.06% mem_cgroup_css_rstat_flush

0.17% blkcg_print_stat

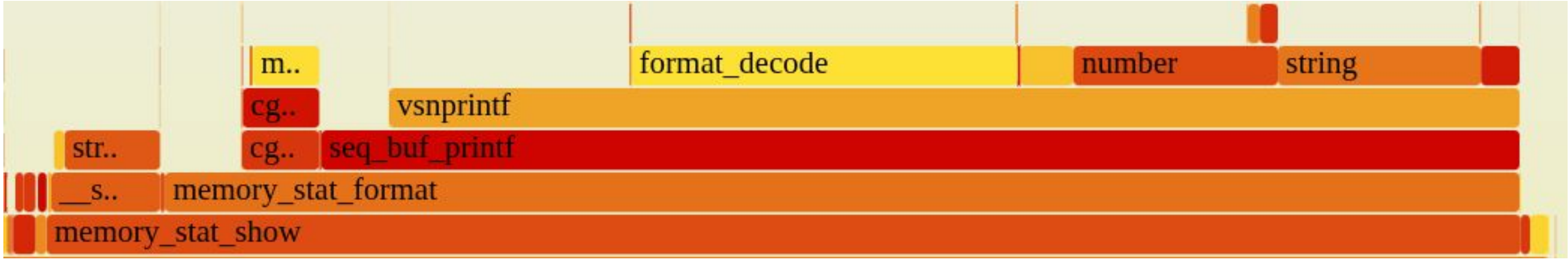
0.11% cpu_stat_show

# Observation

Some call stacks in memory_stat_show stood out:

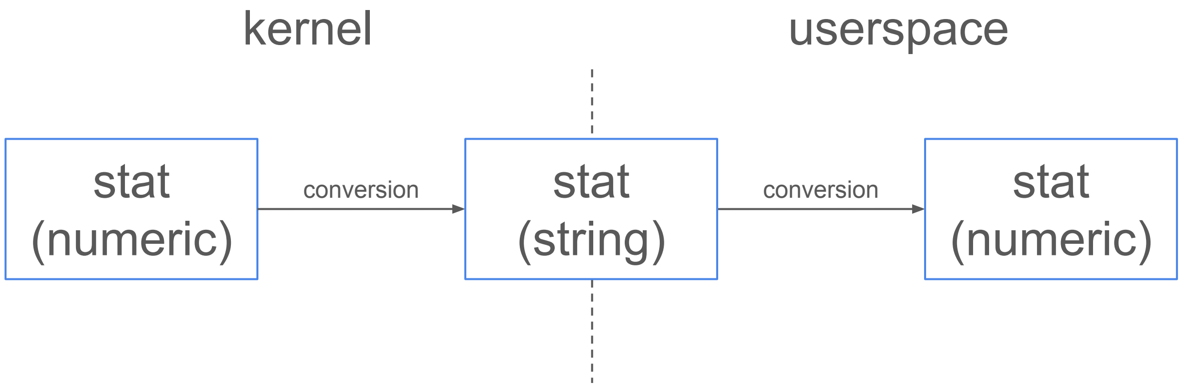



Significant time spent in __seq_puts() and related formatting:



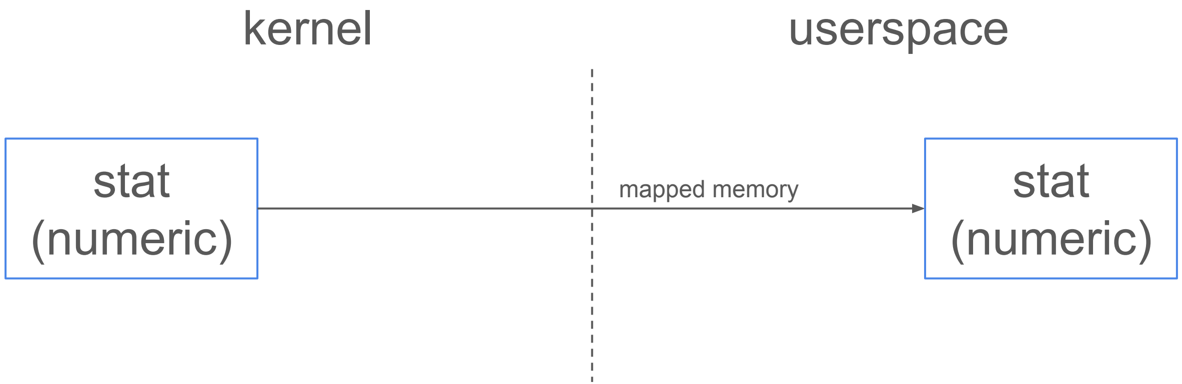
# Can we avoid the overhead of conversion?

Instead of this:



# Can we avoid the overhead of conversion?

How about this:



# A solution using BPF

Foundation: cgroup iterator program

New BPF kfuncs (pending upstream):

- Acquire memcg
- Flush memcg stats
- Fetch stats

Some advantages:

- Conversions eliminated
  - User program can read fields of skel->data_query.memcg_query.
- Parsing/Filtering eliminated
  - BPF program can fetch specific stats.

```
/* The latest values read are stored here. */
struct memcg_query memcg_query SEC(".data.query");

SEC("iter.s/cgroup")
int cgroup_memcg_query(struct bpf_iter__cgroup *ctx)
{
    struct cgroup *cgrp = ctx->cgroup;
    struct cgroup_subsys_state *css;
    struct mem_cgroup *memcg;

    if (!cgrp)
        return 1;

    css = container_of(cgrp, struct cgroup_subsys_state, cgroup);
    if (!css)
        return 1;

    memcg = bpf_get_mem_cgroup(css);
    if (!memcg)
        return 1;

    bpf_mem_cgroup_flush_stats(memcg);

    memcg_query.nr_anon_mapped = bpf_mem_cgroup_page_state(memcg, NR_ANON_MAPPED);
    memcg_query.nr_shmem = bpf_mem_cgroup_page_state(memcg, NR_SHMEM);
    memcg_query.nr_file_pages = bpf_mem_cgroup_page_state(memcg, NR_FILE_PAGES);
    memcg_query.nr_file_mapped = bpf_mem_cgroup_page_state(memcg, NR_FILE_MAPPED);
    memcg_query.memcg_kmem = bpf_mem_cgroup_page_state(memcg, MEMCG_KMEM);
    memcg_query.pgfault = bpf_mem_cgroup_vm_events(memcg, PGFAULT);

    bpf_put_mem_cgroup(memcg);

    return 0;
}
```

# Benchmark - Reading memcg Stats 1M Cycles

***control: elapsed time***

real 0m38.318s

user 0m25.131s

sys 0m13.070s

***experiment: elapsed time***

real 0m2.789s

user 0m0.187s

sys 0m2.512s

***control: perf data***

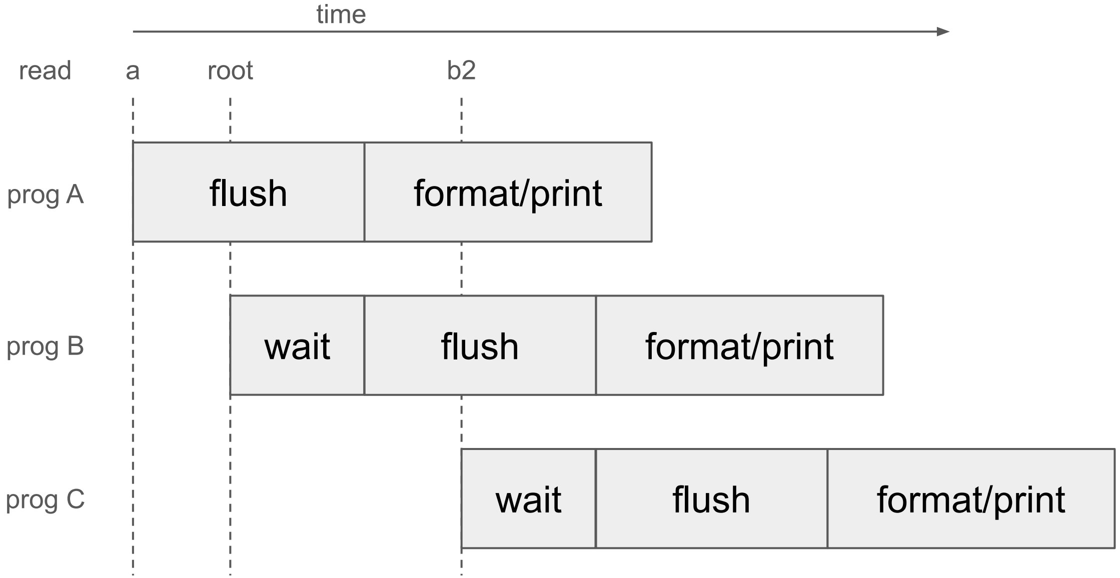
33.43% a.out libc.so.6 [.] __vfscanf_internal
6.88% a.out [kernel.kallsyms] [k] vsnprintf
6.33% a.out libc.so.6 [.] _IO_fgets
5.51% a.out [kernel.kallsyms] [k] format_decode

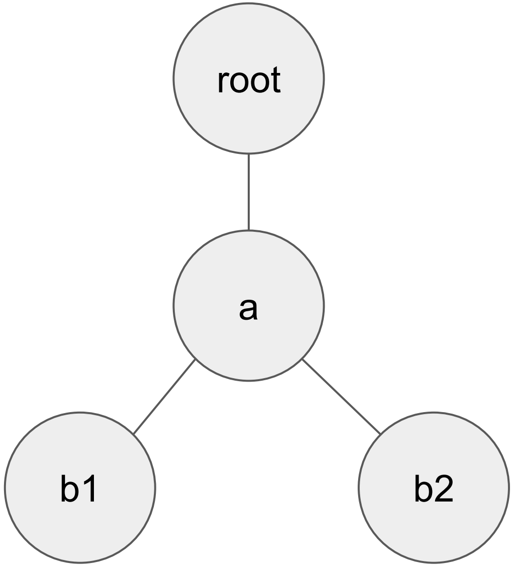
***experiment: perf data***

10.55% memcgstat bpf_prog_..._query [k] bpf_prog_…_query
6.90% memcgstat [kernel.kallsyms] [k] memcg_page_state_output
3.55% memcgstat [kernel.kallsyms] [k] _raw_spin_lock
3.12% memcgstat [kernel.kallsyms] [k] memcg_events

# Case Study: Reader Latency on Random Readers

Multiple programs reading stats can lead to parallel reads which can contend on flushing.

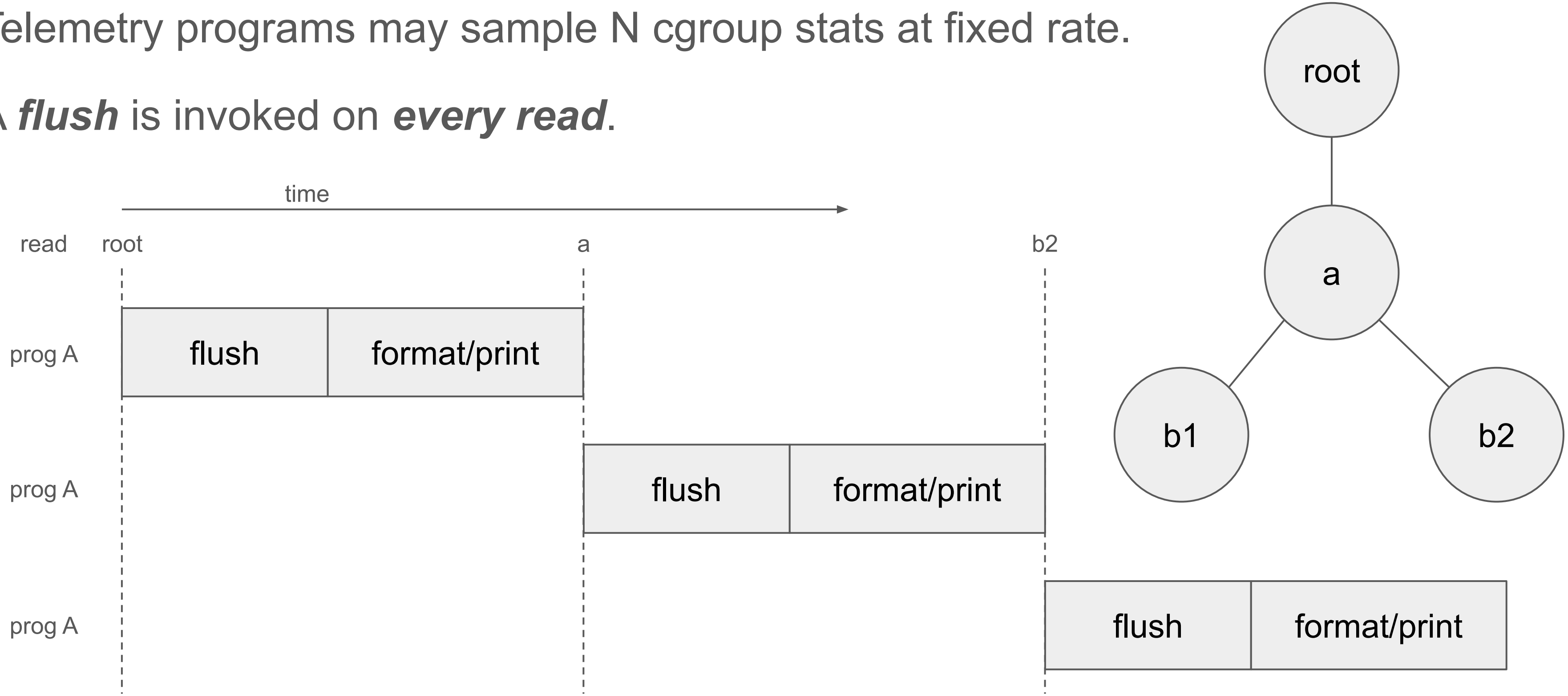


# Case Study: Reader Overhead on Telemetry Program

Telemetry programs may sample N cgroup stats at fixed rate.

A ***flush*** is invoked on ***every read***.

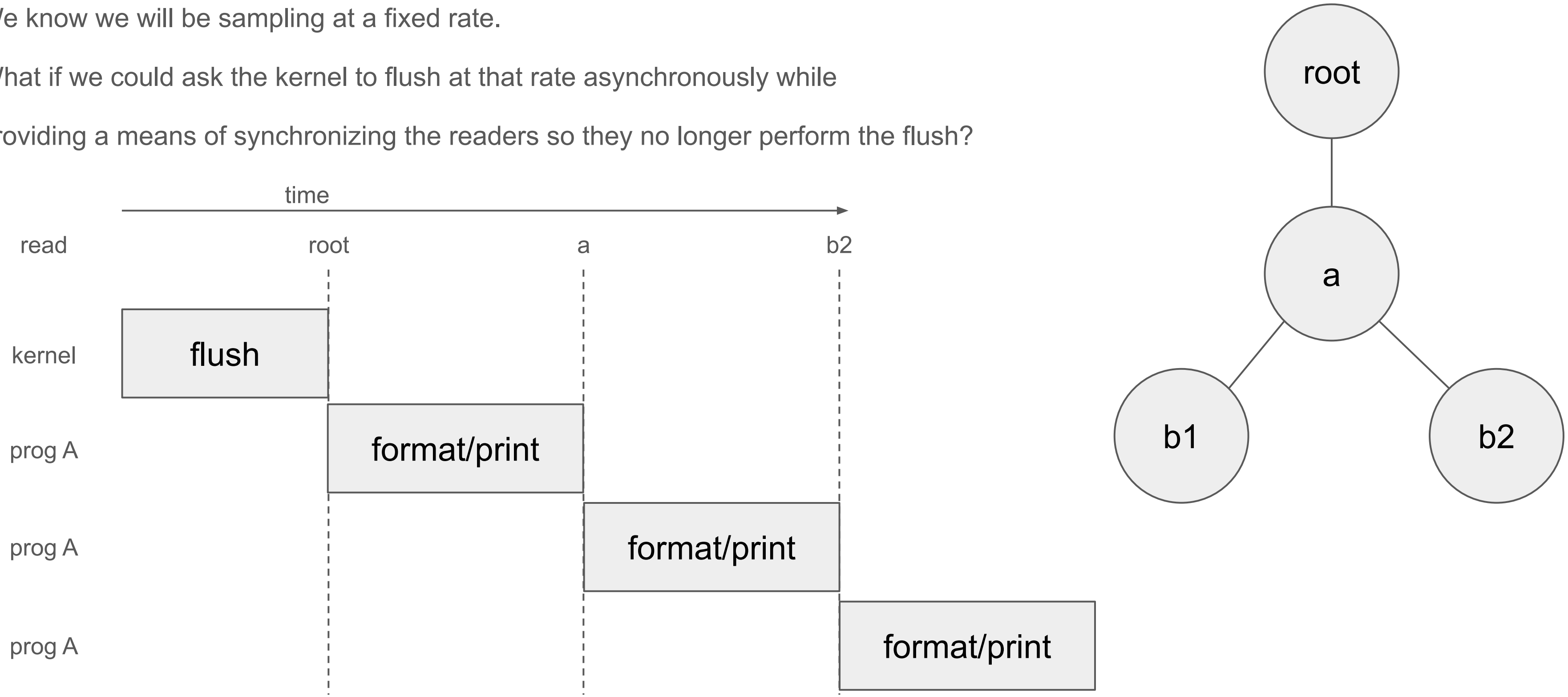


# RFC - Alternative Reading Paradigm

We know we will be sampling at a fixed rate.

What if we could ask the kernel to flush at that rate asynchronously while

providing a means of synchronizing the readers so they no longer perform the flush?



# RFC - Alternative Reading Paradigm

What could this look like?

- Change from read, sleep, read, … to read, poll, read
- Readers can be synchronized with a sequence number and wait/notify action.
- New cgroupfs files
    - /sys/fs/cgroup/cpu.stat.stream_interval (*root cgroup only*)
        - Write interval millisecs value which is used to continuously schedule delayed work (flush stats).
        - After each cycle, a (writer) sequence number is incremented and waiters are notified.
    - /sys/fs/cgroup/cpu.stat.stream
        - Opening sets reader sequence value to zero.
        - Reading invokes wait for event of writer sequence becoming greater than reader sequence.
            - When ready, stats are sent to userspace just as they are now.

Would this help you?

- There were some patches seen in the community that seem to indicate a demand.

# References

Patches landed in 6.16

- Cgroup rstat subsystem separation
  - https://lore.kernel.org/all/20250515001937.219505-1-inwardvessel@gmail.com/

Patches pending upstream (currently part of BPF OOM series)

- Acquire/Release memcg kfuncs
  - https://lore.kernel.org/all/20251027231727.472628-9-roman.gushchin@linux.dev/
- Fetch/Flush memcg stats kfuncs
  - https://lore.kernel.org/all/20251027231727.472628-11-roman.gushchin@linux.dev/
- Selftests for using kfuncs above
  - https://lore.kernel.org/all/20251027232206.473085-2-roman.gushchin@linux.dev/

RFC Patches

- Cgroup stat streaming and reader synchronization
  - Sending after presentation