Samsung Research

Modernizing Resource Management in Embedded Systems using eBPF

Michał Wilczyński, Samsung Electronics

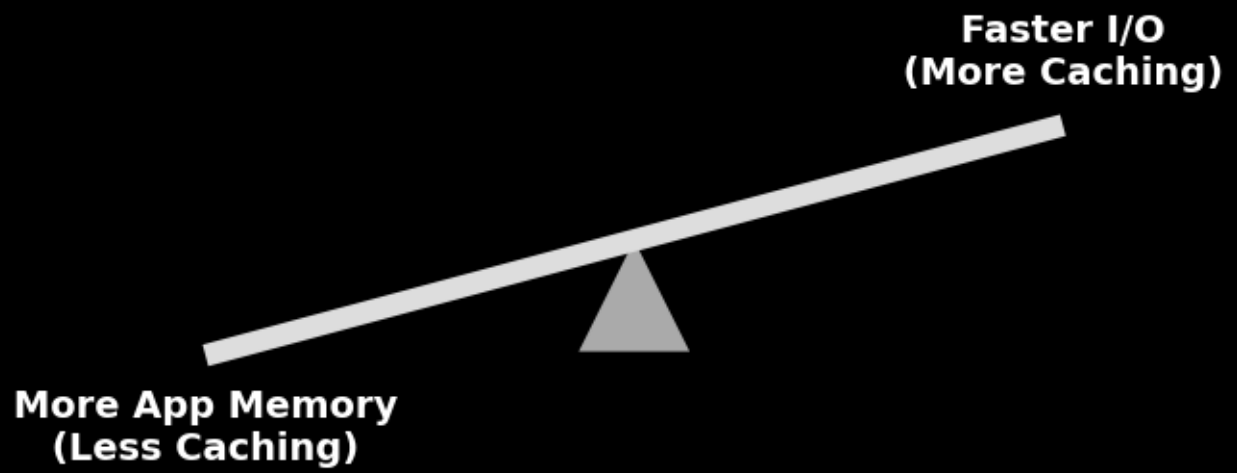
27.08.2025

Agenda

1. The challenge: Resource Management on Embedded Systems
2. Our Approach: Low Overhead Observability with eBPF
3. Solution #1: The proactive Low Memory Monitor
4. Solution #2: The dynamic I/O Cache Tuner
5. Conclusion & Key Takeaways

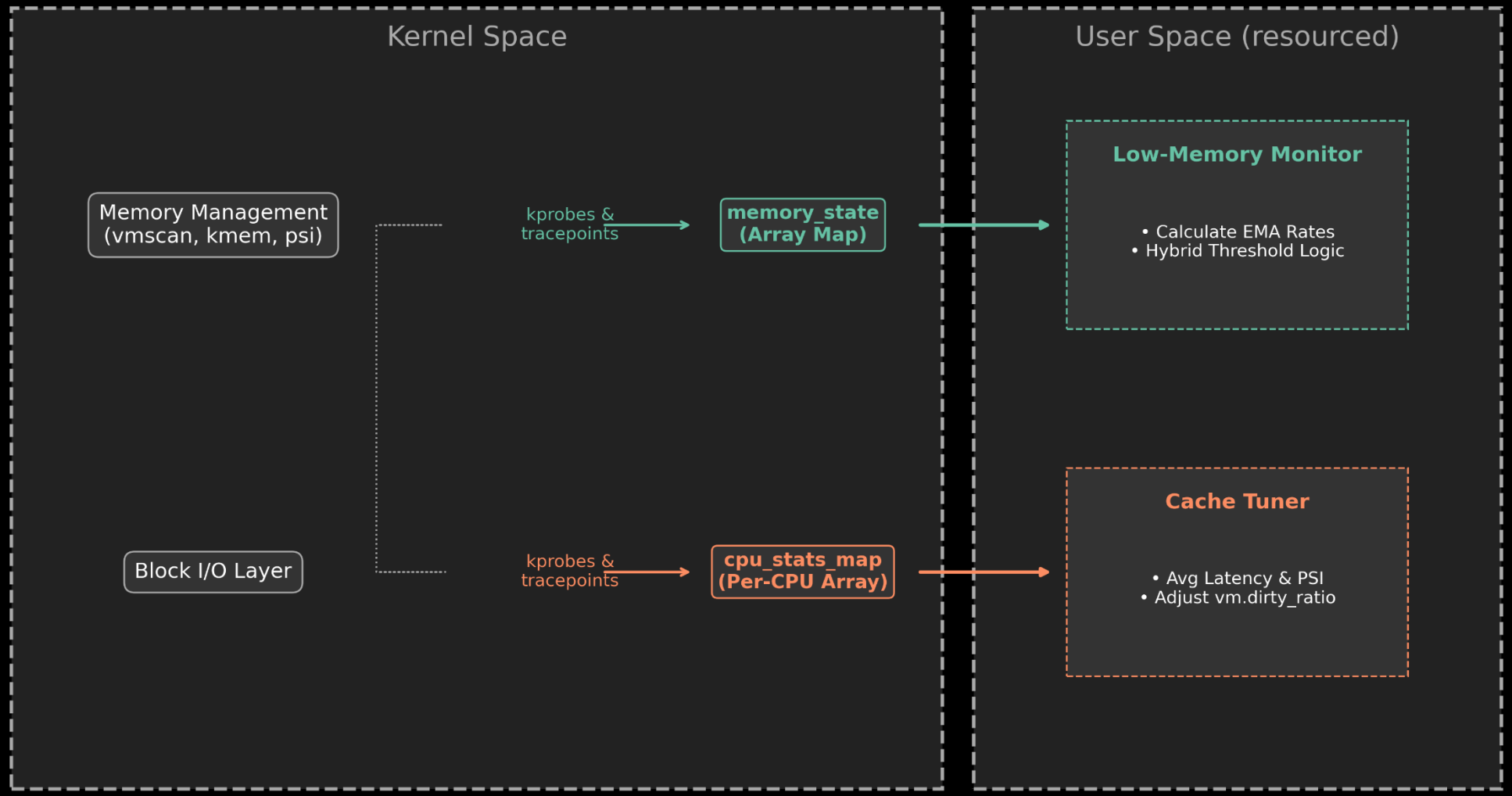
©2025 Samsung Research. All rights reserved

1 The challenge: Resource Management on Embedded Systems



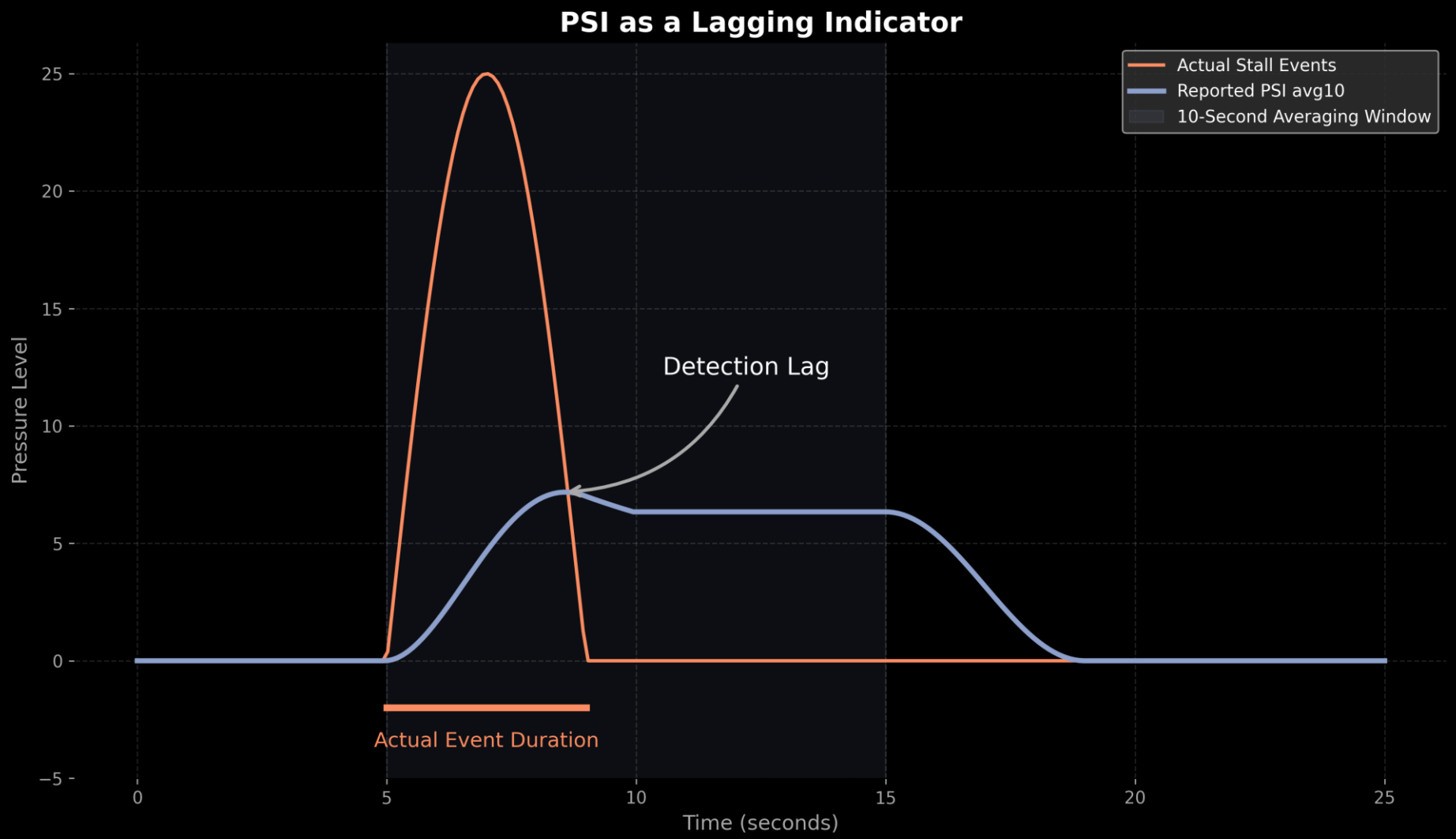
©2025 Samsung Research. All rights reserved

2 Our Approach: Low Overhead Observability with eBPF



©2025 Samsung Research. All rights reserved

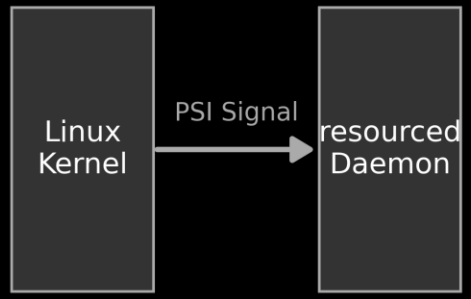
3 Background: What is Pressure Stall Information



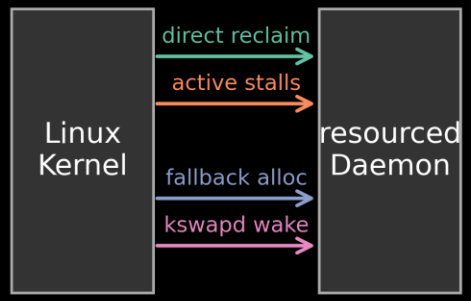
©2025 Samsung Research. All rights reserved

3 Solution #1: The proactive Low Memory Monitor

Traditional Method: One Lagging Signal



Our eBPF Method: Multiple Precise Signals



©2025 Samsung Research. All rights reserved

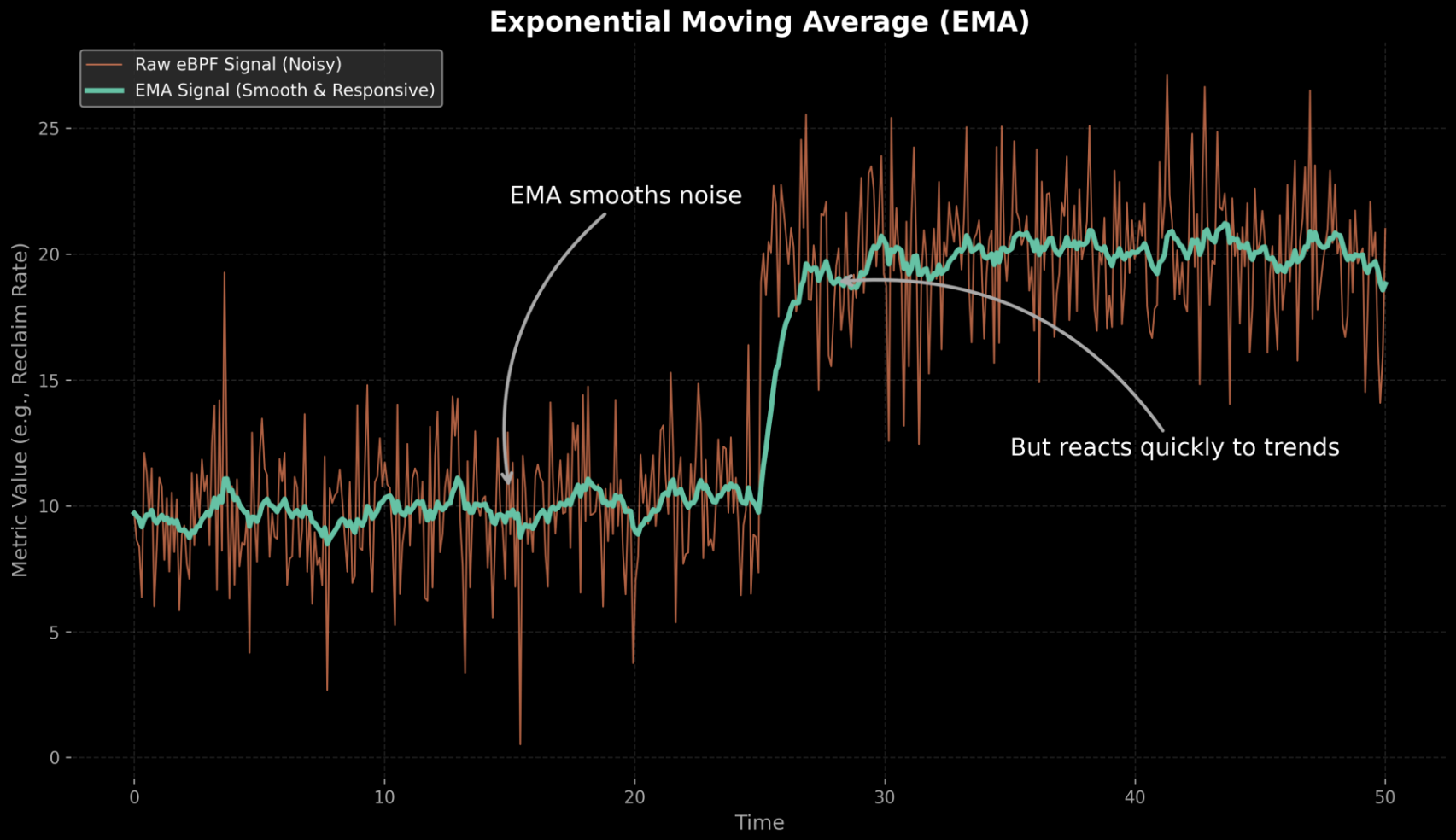
3 Low-Mem Monitor: The eBPF Probes

```
/* BPF map shared with userspace */
struct {
 __uint(type, BPF_MAP_TYPE_ARRAY);
__uint(max_entries, 1);
 __type(key, __u32);
 __type(value, struct memory_state)
;} memory_state SEC(".maps");
```

```
/* Tracepoint on direct memory reclaim */
SEC("tracepoint/vmscan/mm_vmscan_direct_reclaim_begin")
int trace_direct_reclaim_begin(...)
{
        struct memory_state *ms = get_memory_state();
        if (!ms)
                return 0;
        /* Atomically increment the counter */
        __sync_fetch_and_add(&ms->direct_reclaim_count, 1);

        return 0;
}
```


©2025 Samsung Research. All rights reserved

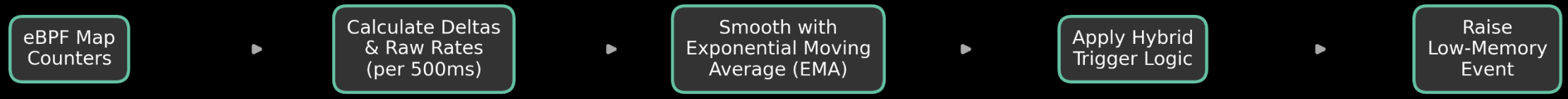
3 Background: What is Exponential Moving Average ?



$$EMA_{new} = (\alpha \cdot V_{current}) + (1 - \alpha) \cdot EMA_{old}$$

©2025 Samsung Research. All rights reserved

3 Low-Mem Monitor: User-space Logic



Userspace polls the eBPF map every 500ms to get raw counter data.

The raw rates are smoothed using an Exponential Moving Average (EMA) to prevent false alarms from brief, insignificant spikes.

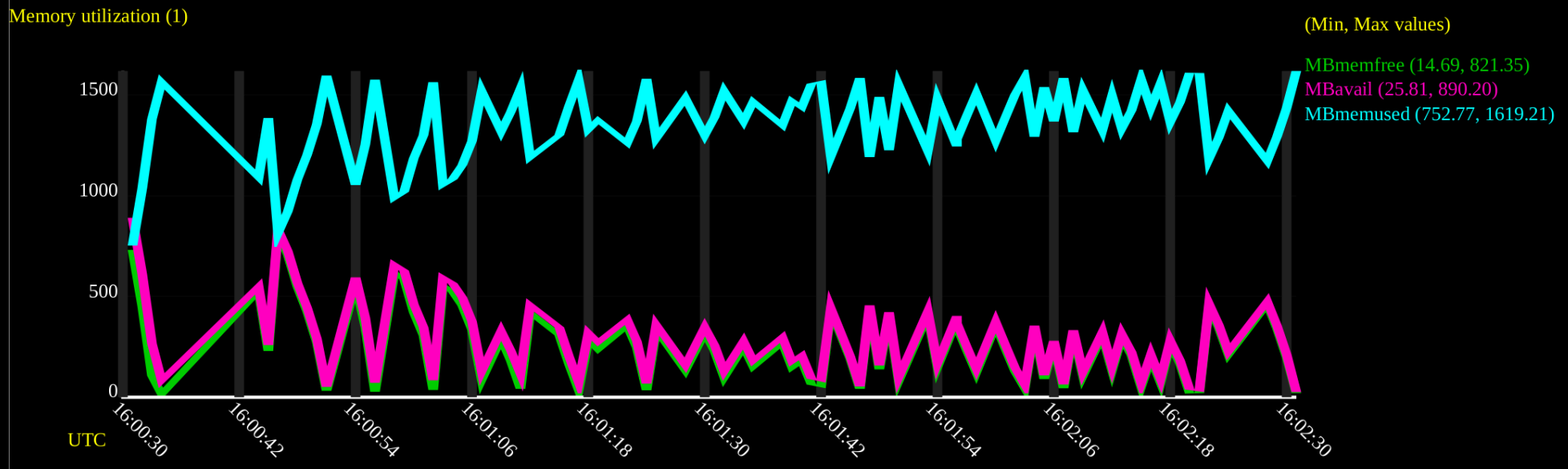
The core of the logic is a hybrid trigger that fires if the smoothed rate shows a rapid relative increase OR if it crosses a high absolute threshold.

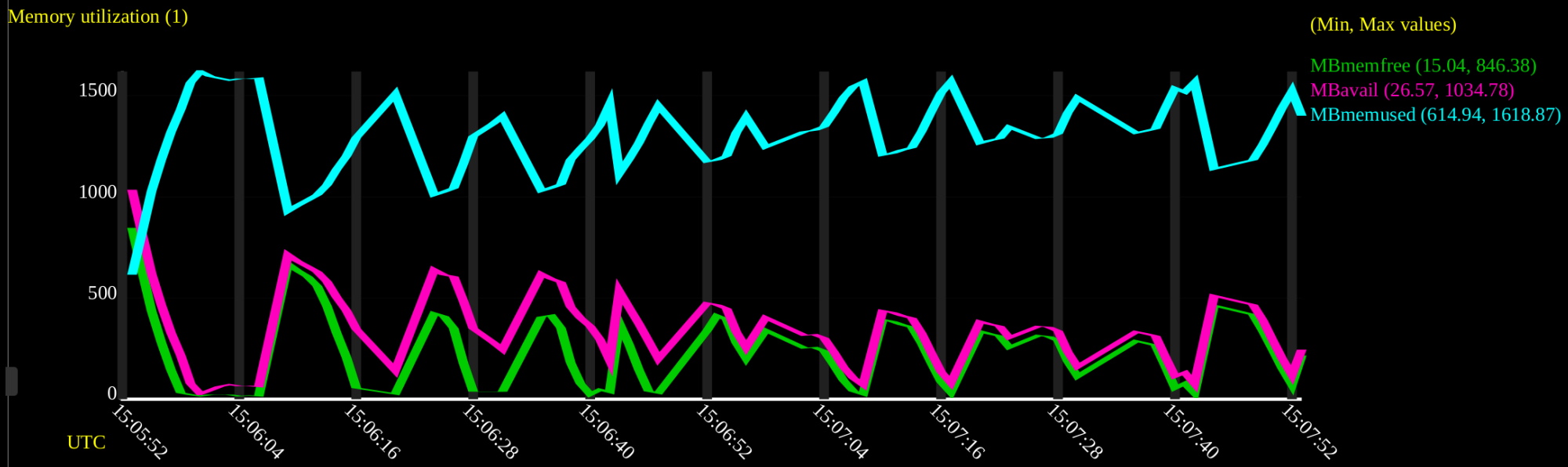
©2025 Samsung Research. All rights reserved

3 Integration with resourced controller

©2025 Samsung Research. All rights reserved

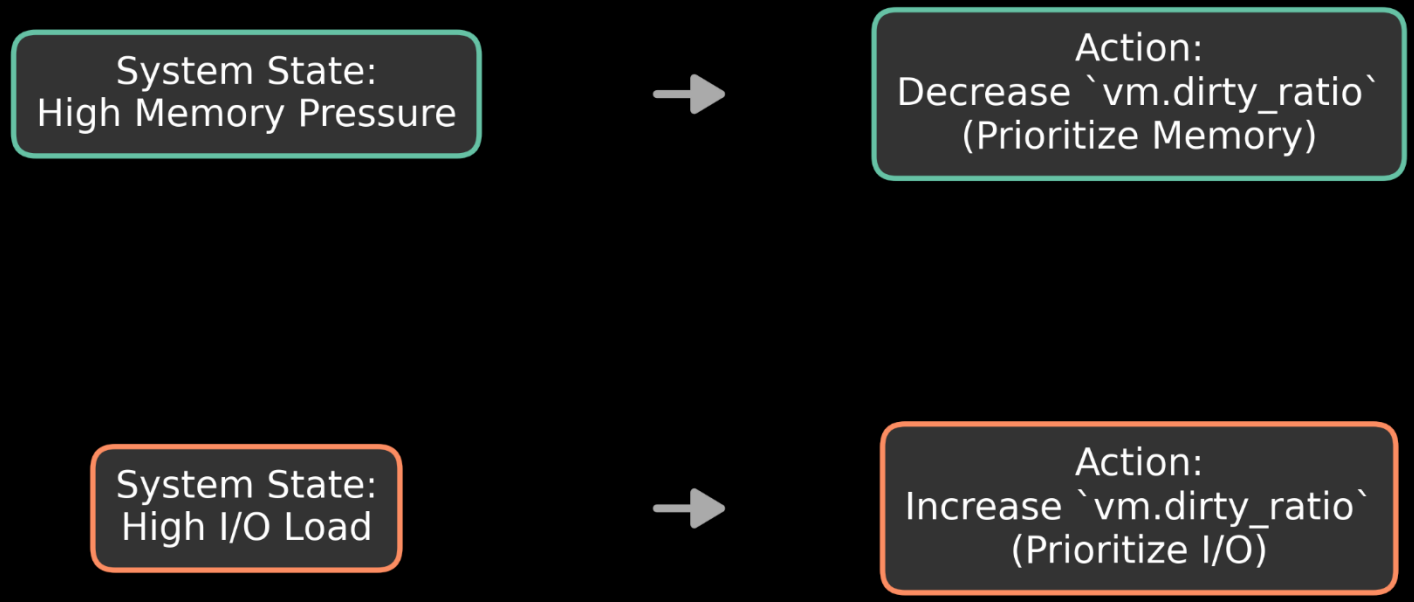
3 Low-mem Monitor: Benchmark results





©2025 Samsung Research. All rights reserved

4 Solution #2: The dynamic I/O Cache Tuner



©2025 Samsung Research. All rights reserved

4 eBPF in Detail – Measuring Memory Reclaim Latency

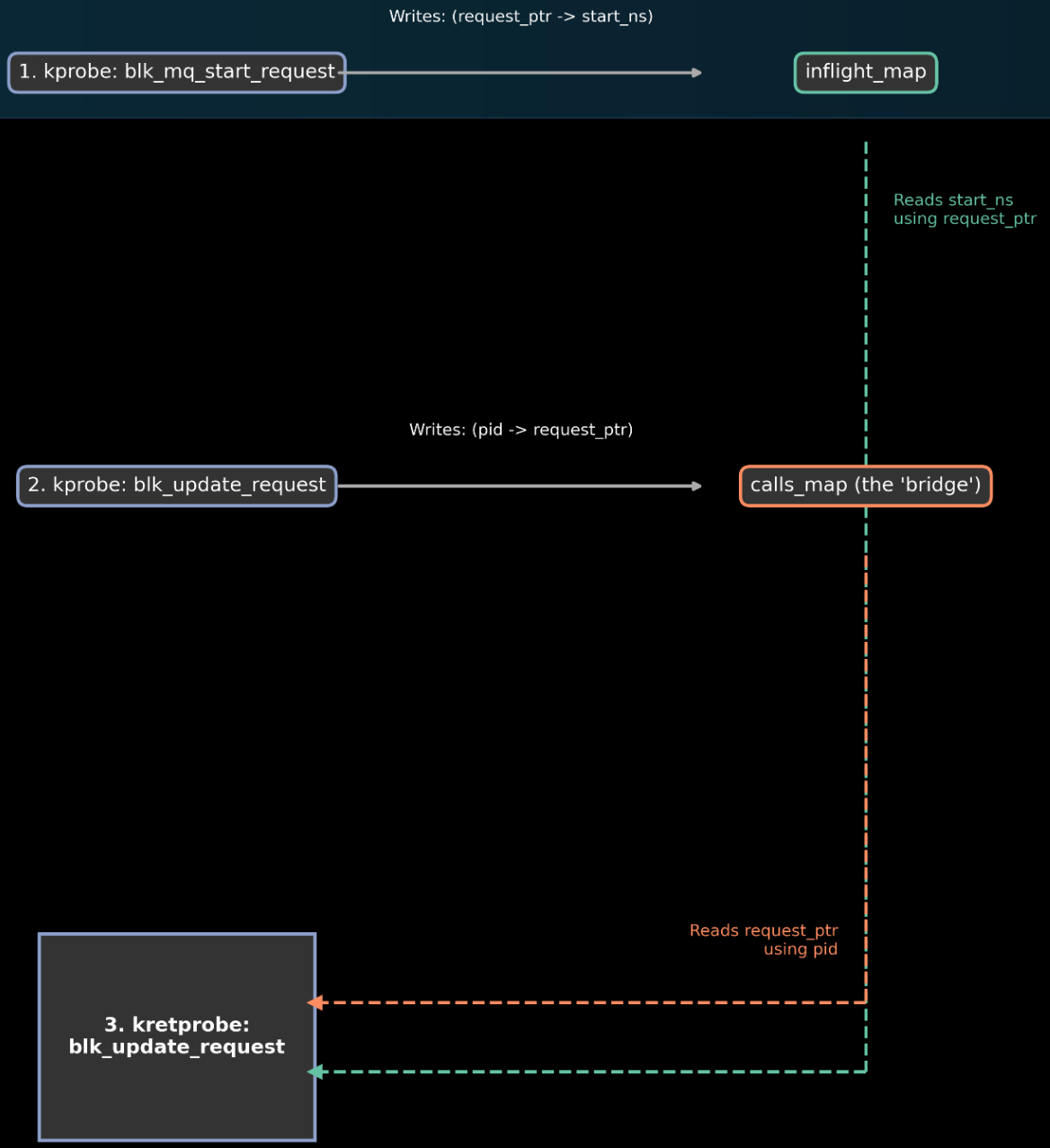
```
/* Store start timestamp on 'begin' if sampled */

SEC("tracepoint/vmscan/mm_vmscan_direct_reclaim_begin")
int handle_direct_reclaim_begin(...)
{
        if (!sample_event())
                return 0;
         // Get per-CPU map entry
        struct dirty_tuner_stats *stats = ...;
        if (!stats)
                return 0;
        stats->in_reclaim_start = bpf_ktime_get_ns();
        return 0;
}
```

```
/* On 'end', calculate duration if we have a start time */

SEC("tracepoint/vmscan/mm_vmscan_direct_reclaim_end")
int handle_direct_reclaim_end(...)
{
        struct dirty_tuner_stats *stats = ...;
        if (!stats || stats->in_reclaim_start == 0)
        return 0;
         __u64 delta = bpf_ktime_get_ns() - stats>in_reclaim_start;
        stats->direct_reclaim_samples += 1;
        stats->direct_reclaim_ns += delta;
        stats->in_reclaim_start = 0;
        /* Reset for next sample */
        return 0;
}
```


©2025 Samsung Research. All rights reserved

4 eBPF in Detail – Measuring I/O Latency

```
/* inflight_map: up to 32 rq_ptr => start_ns */
struct {
        __uint(type, BPF_MAP_TYPE_HASH);
        __uint(max_entries, 32);
        __type(key, __u64);
        __type(value, __u64); /* storing start_ns */
} inflight_map SEC(".maps");

/* calls_map: up to 32 pid_tgid => rq_ptr * (for the kretprobe
bridging) */
struct {
        __uint(type, BPF_MAP_TYPE_HASH);
        __uint(max_entries, 32);
        __type(key, __u64);
        __type(value, __u64);
} calls_map SEC(".maps");
```



©2025 Samsung Research. All rights reserved

4 eBPF in Detail – Measuring I/O Latency

```
/* kretprobe fires after the I/O completion function returns */
SEC("kretprobe/blk_update_request")
int BPF_KRETPROBE(ret_blk_update_request)
{
        // Was this the final completion?
        bool more = (bool)PT_REGS_RC(ctx);
        if (more)
                return 0; // Not done yet, ignore partial completion
        // 1. Get the original request pointer from our "bridge" map
        __u64 pid_tgid = bpf_get_current_pid_tgid();
        __u64 *rq_ptr_p = bpf_map_lookup_elem(&calls_map, &pid_tgid);
        if (!rq_ptr_p)
                return 0;
        // 2. Get the start time from the inflight map
        __u64 *start_ns_p = bpf_map_lookup_elem(&inflight_map, rq_ptr_p);
        if (!start_ns_p)
                return 0;
        // 3. Calculate latency and update stats
        __u64 delta = bpf_ktime_get_ns() - *start_ns_p;
        // ... update cpu_stats_map ...
        // 4. Clean up map entries to prevent leaks
        bpf_map_delete_elem(&calls_map, &pid_tgid);
        bpf_map_delete_elem(&inflight_map, rq_ptr_p);
        return 0;
}
```


©2025 Samsung Research. All rights reserved

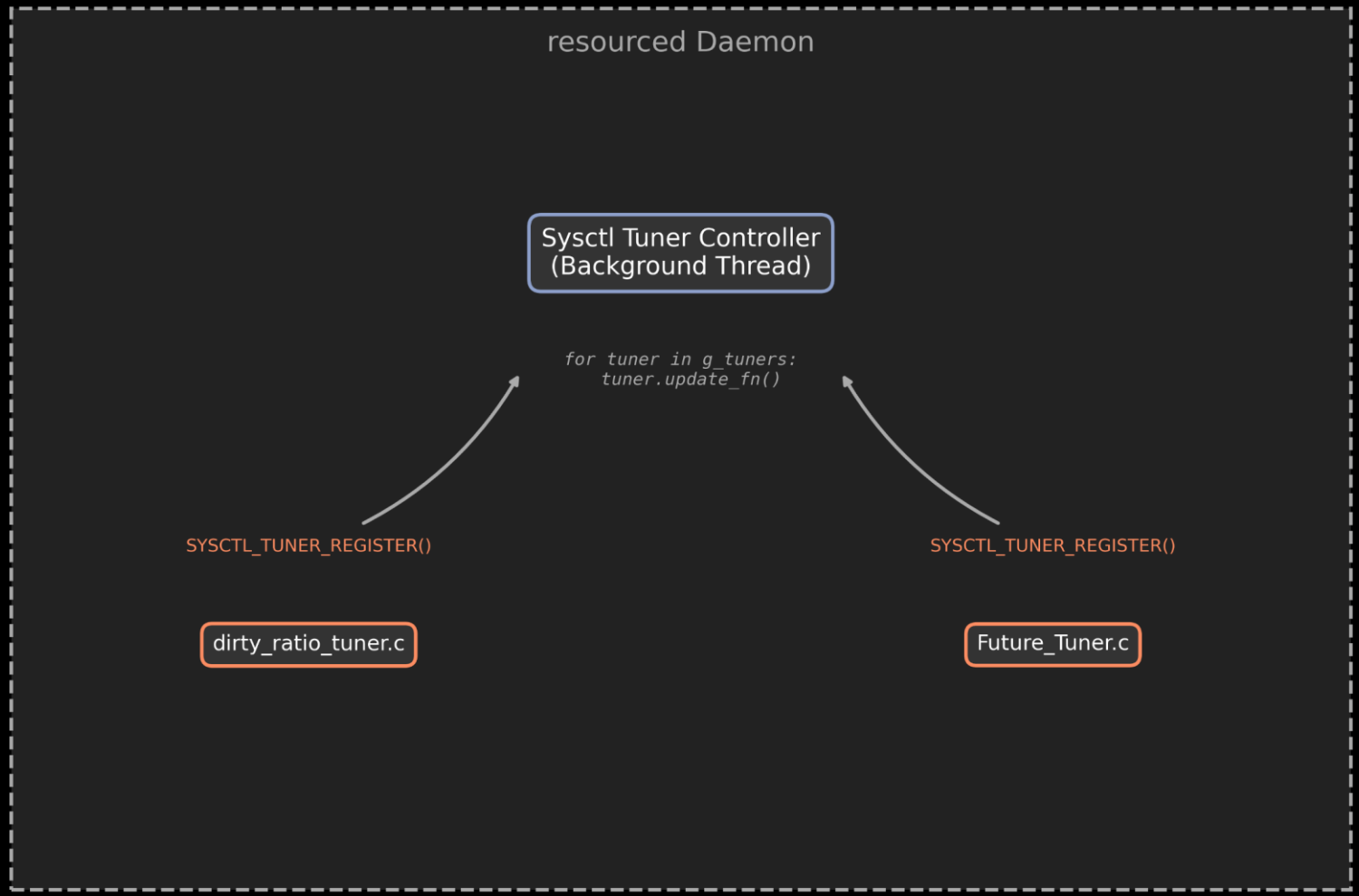
4 Cache Tuner: An Extensible Framework

```
/* Each tuner must implement this interface */
struct sysctl_tuner_ops {
    const char *name;
    int (*init_fn)(struct bpf_skel_info *info);
    int (*exit_fn)(struct bpf_skel_info *info);
    void (*update_fn)(struct bpf_skel_info *info);
};
```

Generic, extensible framework within resourced to manage multiple tuners.

Each tuner is a self contained module that implements a standard sysctl_tuner_ops interface with init, exit, and update functions.

Single background thread iterates through all registered tuners and calls their update function periodically.



©2025 Samsung Research. All rights reserved

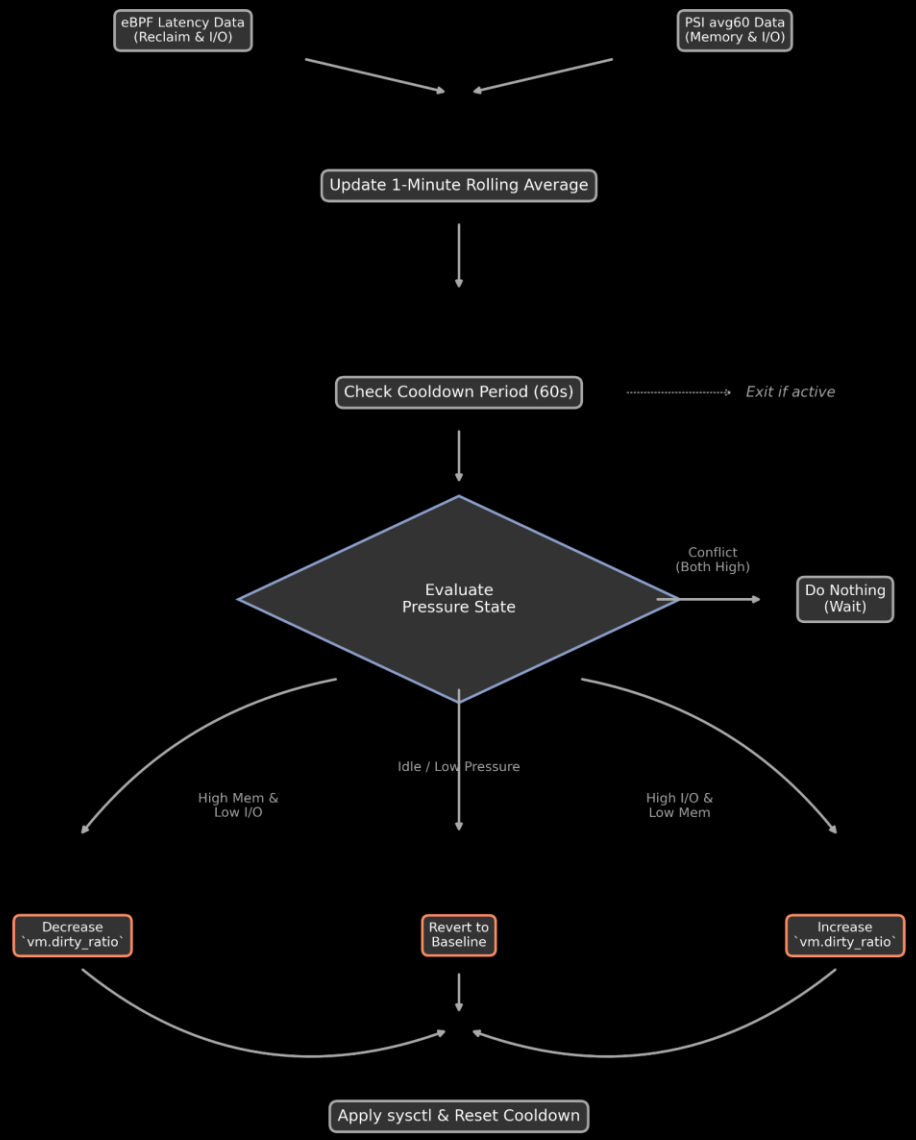
4 Cache Tuner: The Decision Logic

The tuner uses a hybrid of data sources, combining high-resolution eBPF latency measurements with long term PSI avg60 data for a complete view of system pressure.

To ensure stability and avoid reacting to brief spikes, the logic averages the eBPF latency data over a one-minute rolling window.

A 60-second cooldown period is enforced after every adjustment to prevent rapid, unstable changes to system parameters ("flapping").

The core logic is a multi-state decision tree: it increases vm.dirty_ratio for high I/O load, decreases it for high memory pressure, and reverts to the original baseline when the system is idle.



©2025 Samsung Research. All rights reserved

4 Cache Tuner: Benchmark Results

Benchmark method

Workload: A mixed I/O and memory stress test using stress-ng.

Details: 4 workers performing 1GB random writes, plus 2 workers allocating 1GB of memory.

Duration: The test ran for 348-411 seconds (~6-7 minutes).

Performance & Efficiency Gains

The tuner reduced benchmark completion time by **15.3%.**

Write throughput saw a massive **46.7%** improvement.

Sustained I/O bandwidth increased by **12.8%.**

The tuner also reduced I/O pressure and used the memory cache more efficiently.

©2025 Samsung Research. All rights reserved

5 Conclusion & Key Takeaways

eBPF provides powerful, low-overhead observability ideal for building intelligent resource managers.

Proactive low-memory monitor prevents system saturation by creating a healthy "saw-tooth" pattern of memory usage, proving its stability and effectiveness.

The dynamic cache tuner delivers significant I/O performance gains, improving write throughput by up to 46.7%.

The architectural pattern (eBPF for data, userspace for logic) is a reusable and effective design.

Github link for code (clone of the gerrit repository): https://github.com/mwilczy/resourced

©2025 Samsung Research. All rights reserved

Thank You!