December/2025

Meta

Resource Management on Embedded Linux Gaming Devices

Stefan Bossbaly
sboss <at> meta.com

Linux Plumbers Conference 2025

Meta

Agenda

1. Why Resource Management is Important
2. What Mechanisms Linux Gives Us
3. How Android Handles Resource Management
4. Expanding on that implementation with Orchestrator
5. Our Success
6. Our Plans for the Future

Meta

Why is Resource Management Important

- Embedded devices are limited in their capabilities
  - Hardware
  - Thermals
  - Energy
- Embedded devices are generally multi purpose
  - Noisy Neighbors
  - Mixed criticality workloads
- Cost Optimization
  - Hardware is expensive
  - More devices to more people

Meta

What Are Our Goals?

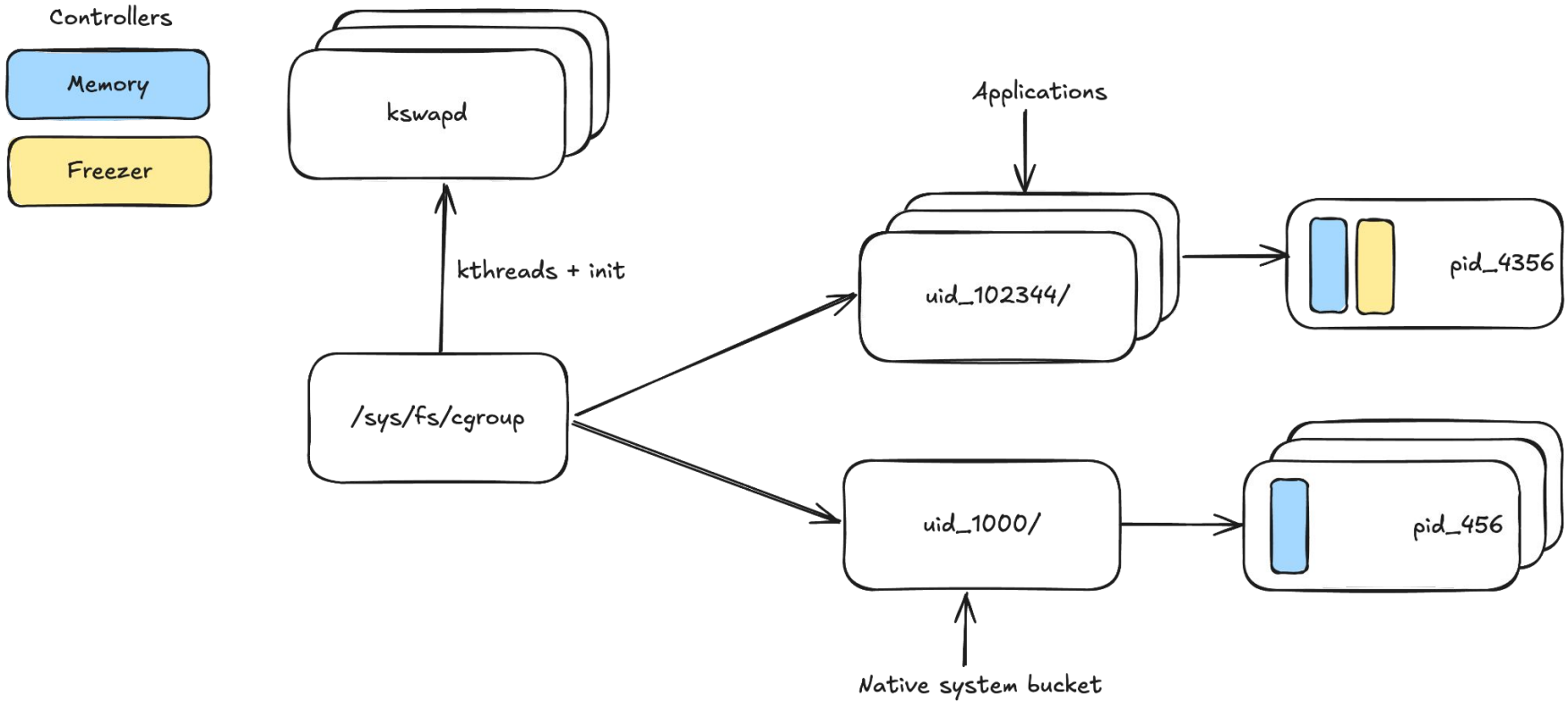
- Accurate collection of resource usage
  - Attribute to a process
  - Low overhead
- Way to control the resource usage per process
- Sole Decision Maker
- Telemetry
  - Log resource usage, detect regressions
  - Log enforcement actions
- Experimentation
  - Trade-offs

Meta

Linux Provided Mechanisms

- Cgroupv2
  - Introduced unified hierarchy over cgroupv1
    - Up to the implementation to organise processes
  - Provides control “knobs” and read-only files that are generic across all devices
  - Controllers are hardware agnostic
- BPF
  - Allows the running of user-specified code at known hook points
  - Aligns perfectly with resource monitoring
  - Can be driver specific, adding “non-standard” hook points
  - Up to the implementation to ensure correctness

Meta


Controllers
Memory
Freezer
kswapd
kthreads + init
/sys/fs/cgroup
Applications
uid_102344/
pid_4356
uid_1000/
pid_456
Native system bucket

Meta

How does Android Controls the Knobs

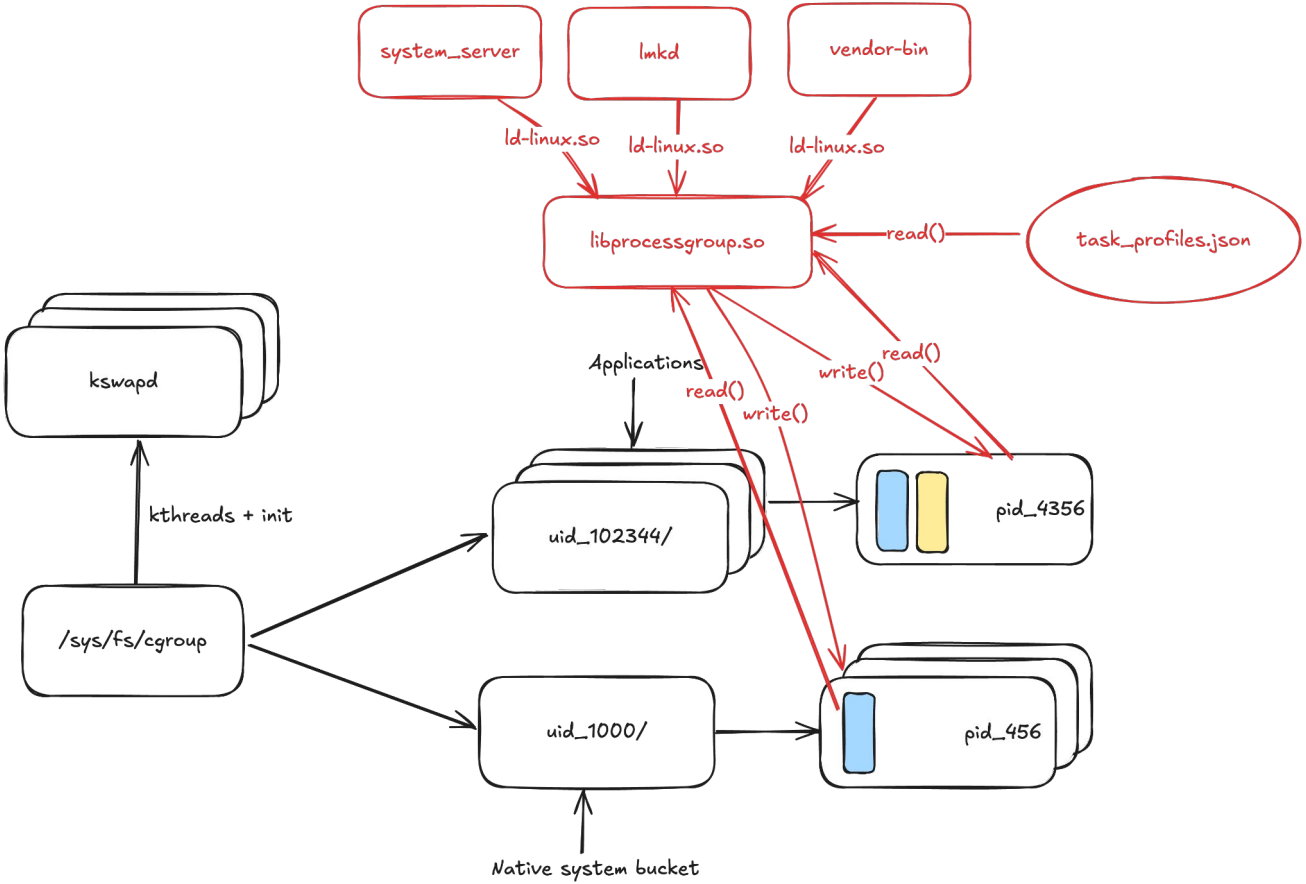
- Uses libprocressgroup, a cgroup abstraction layer
- Source of truth: cgroups.json & task_profiles.json
- Terminology
  - Attributes
  - Actions
  - Profiles
  - Aggregate Profiles
- Profile and Aggregate Profiles can then be applied to processes
- system_server applies resource for application
- Native Services apply them to themselves

Meta

Example task_profiles.json

```
{
  "Attributes": [
    { "Name": "MemcgHigh", "Controller": "memory", "File": "memory.high" },
    { "Name": "MemcgMax", "Controller": "memory", "File": "memory.limit_in_bytes", "FileV2": "memory.max" }
  ],
  "Profiles": [
    {
      "Name": "SCHED_SP_BACKGROUND",
      "Actions": [
        { "Name": "SetAttribute", "Params": { "Name": "MemcgHigh", "Value": "128MB" } },
        { "Name": "SetAttribute", "Params": { "Name": "MemcgMax", "Value": "256MB" } }
        { “Name”: “JoinCgroup”, “Params”: { Controller": "cpu", “Path”: “background” } }
      ]
    }
  ]
}
```

Meta

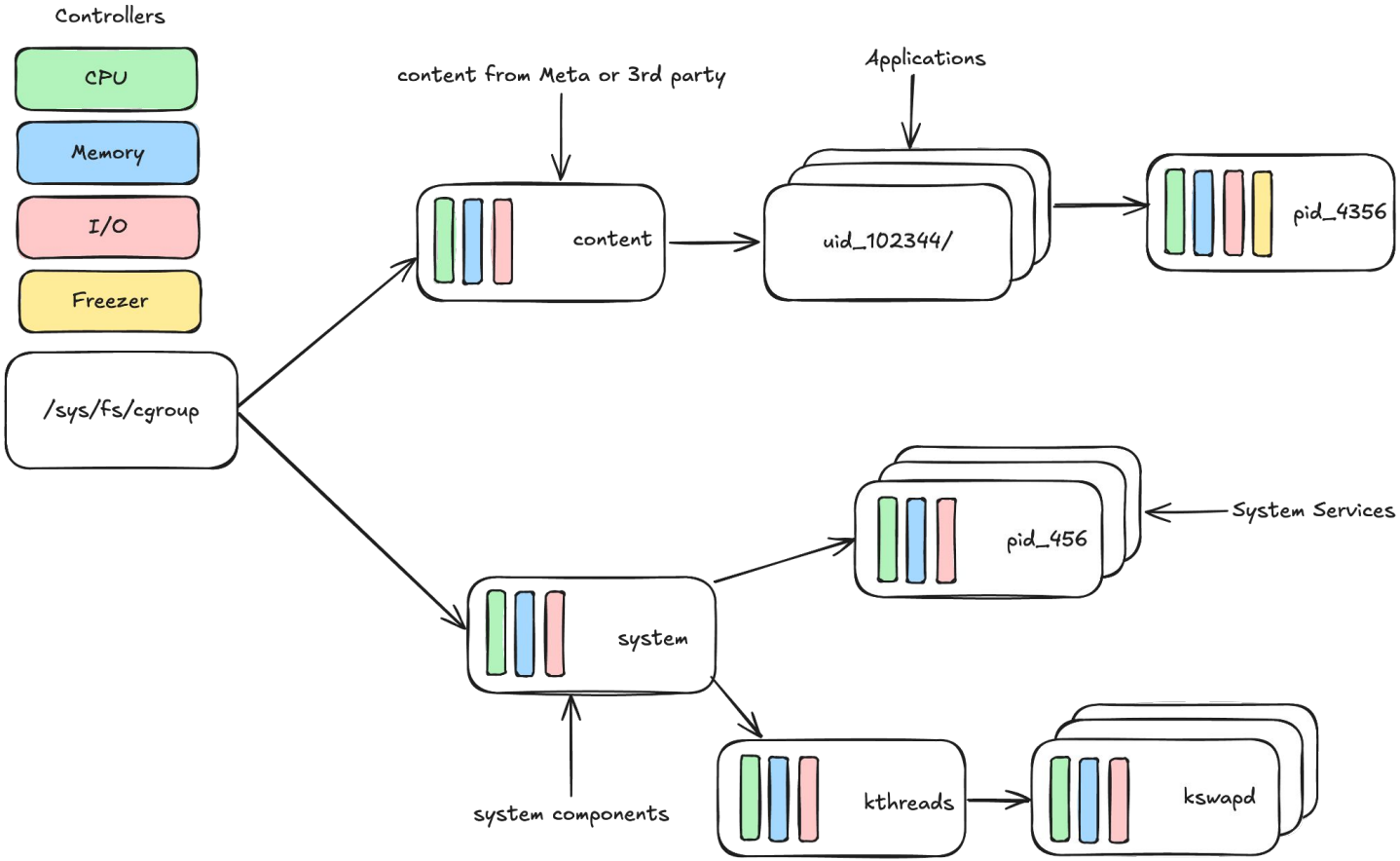

system_server
lmkd
vendor-bin
ld-linux.so
ld-linux.so
ld-linux.so
libprocessgroup.so
read()
task_profiles.json
kswapd
Applications
read()
write()
read()
write()
kthreads + init
uid_102344/
pid_4356
/sys/fs/cgroup
uid_1000/
pid_456
Native system bucket

Meta

Expanding libprocessgroup

- Benefits
  - Cgroup Abstraction
  - Centralizes Configuration
  - Vendor Overriding
- Limitations
  - Cgroup abstraction is incomplete
    - cgroupv1 cpuctl implementation
- Resource Management as shared library
  - Deconfliction is nearly impossible
- Experimentation
  - A/B Dynamic Configuration

Meta


Controllers
CPU
Memory
I/O
Freezer
/sys/fs/cgroup
content from Meta or 3rd party
content
Applications
uid_102344/
pid_4356
pid_456
System Services
system
system components
kthreads
kswapd

Meta

Implementing a Centralized Approach to Resource Control

- Rust based policy framework called Orchestrator
- Solely responsible for the management of cgroup hierarchy
- Components
  - Cgroupv2 controller interface
  - Policy Manager and Executor
  - BPF Programs for Metric Collection
  - RPC interface for applications to provide context
- Goals
  - Policy Framework with reusable components
  - Unify the cgroup management into one process
  - Unification of resource monitoring and resource management

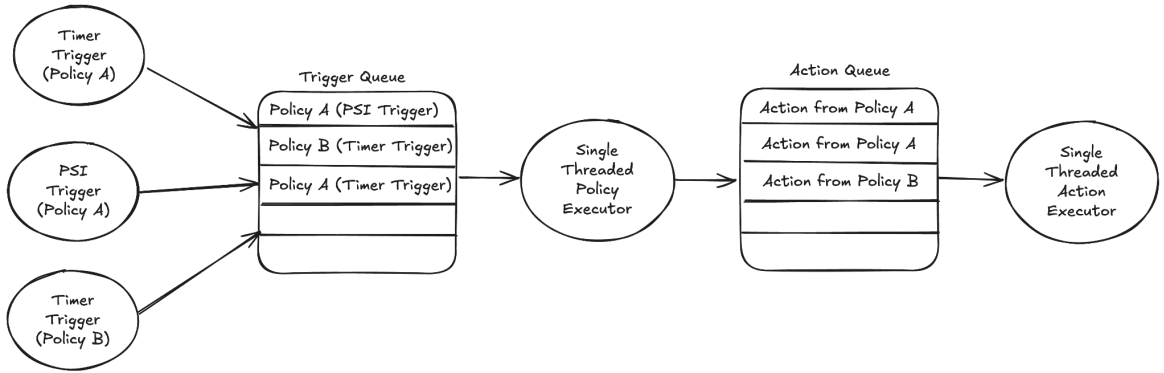
Meta


External Service
RPC Interface
Resource Measurements
BPF Programs
Policies
State
Actions
orchestrator
read/write/epoll_ctl
read/write/epoll_ctl
read/write/epoll_ctl
read/write/epoll_ctl
content
uid_102344/
pid_4356
/sys/fs/cgroup
read/write/epoll_ctl
read/write/epoll_ctl
pid_456
system
kthreads
kswapd

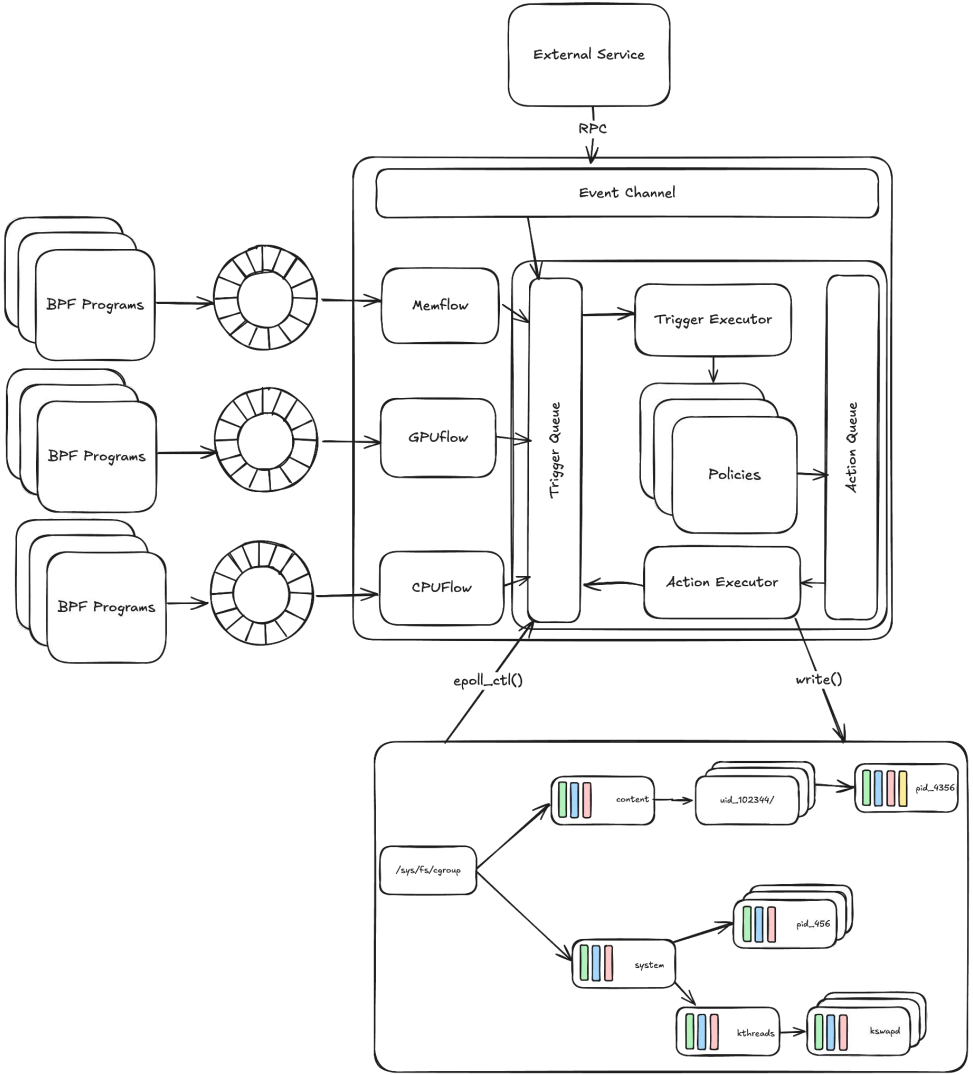
Meta

Policy Based Engine

- Terminology
  - Policy
  - Policy Factory
  - Trigger
  - Action
- Framework Benefits
  - Real-time guarantee
  - Clear Responsibilities
  - Deconflicting
  - Experimentation



Meta


External Service
RPC
Event Channel
BPF Programs
BPF Programs
BPF Programs
Memflow
GPUflow
CPUFlow
Trigger Queue
Trigger Executor
Policies
Action Queue
Action Executor
epoll_ctl()
write()
/sys/fs/cgroup
content
uid_102344/
pid_4356
system
pid_456
kthreads
kswapd

Meta

```
define_policy! {
    MemoryPolicy {
        description: "A test policy to show case the define_policy! schemantics",
        state: {
            count: u32,
        },
        triggers: {
            (PsiHigh: PsiTrigger),
            (PsiLow: PsiTrigger),
            (Timer: TimerTrigger),
        },
        queue: Some(Queue::Memory),
        execute: |trigger_event| {
            match trigger_event {
                Self::PsiHigh(_psi_data) => {
                    Some(Action::KillLowPriorityProcesses)
                }
                Self::PsiLow(_psi_data) => {
                    self.count += 1;
                    None
                }
                Self::Timer(duration) => {
                    let current_count = self.count;
                    self.count = 0;
                    if current_count > 10 {
                        Some(Action::PerformMemoryAudit)
                    } else {
                        None
                    }
                }
            }
        }
    }
}
```

Meta

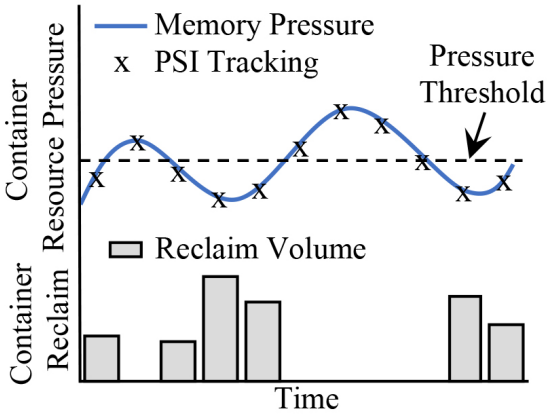
Currently Deployed at Meta

- Orchestrator is still in its infancy
- Memory resource is first to be implemented
- Telemetry Policy
  - Provides high level view of memory usage per process
  - Leverages BPF provided data
- Heap Type Telemetry Policy
  - Provides detailed view of heap type allocations per process
- Budget Service Policy
  - Leverages cgroupv2 knobs
  - Tracks memory for all sources
  - Solves the noisy neighbor problem

Meta

The Future

- Transparent Memory Offloading Policy
  - Swapping of unused pages
  - Pressure Stall Information
- CPU Policies
  - Budget Policy
- GPU Policies
  - Budget Policy
- Open Sourced
  - Github



Source: Engineering at Meta

Meta

Thank you.