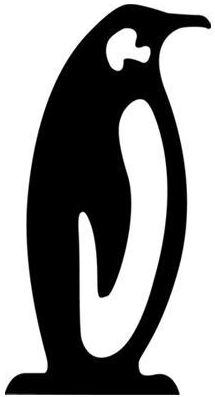


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Mempolicy Is Dead
Long Live Memory Policy

Gregory Price

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

What's the story here?

"I have special memory and I don't want to use it like memory. I want malloc()."
~ Many $DEVICE_DEV for many $REASON.

"Don't put the memory in the page allocator…"
~ Many $KERNEL_DEV for many $REASON.
(good $KERNEL_DEV, good $REASON)

Corollary: "also you must re-implement everything in mm/"

What do we do instead?

- DAX
  - The X shaped device goes in the DAX shaped hole
  - A decade later: Promise of userland allocators never materialized
- HMM / ZONE_DEVICE migration
  - complexity, overhead - still limited core services
- **People shove it in the page allocator anyway**
  - Platform can just set EFI_CONVENTIONAL_MEMORY
  - memory as NUMA node in ZONE_NORMAL (worst case)
  - page_alloc, reclaim, compaction end up using this to poor effect.

What are the use cases?

For some classes of memory, want mm/ services w/ alloc & access controls

- GPU / Accel memory
  - Don’t want spillage.
  - Might want reclaim / compaction / PT controls
- Compressed Memory device
  - Want really hard write-access controls to prevent fatal OOM
  - Want most mm/ services
- Slow (X00ns+) / Contended BW Memory (Large memory pool)
  - Don’t want in the fallback list.
  - Do want demotion and promotion.
  - Do want reclaim, compactions, probably hotplug.

What does [in the page allocator] mean?

Object — member of —> Object

Page —> Memory Zone
Zones —> Memory Node
Memory Node —> node_states[N_MEMORY]

page_alloc **default** memory policy (nid=NUMA_NO_NODE, nodemask=NULL):
(zone < DEVICE) for node in node_states[N_MEMORY]

Take away: in the page allocator = in N_MEMORY

What about existing policy mechanisms?

cgroup.cpusets.mems_allowed

- allows you to restrict node access for tasks in a cgroup hierarchically
- if cpuset.mems=[0], those tasks can only access node 0.
- Removing node from root.mems prevents **everything*** from accessing a node.
- Not isolating the root means root tasks may access the node. **Insufficient as-is**.

mempolicy

- completely disregarded by a variety of mm/ services (reclaim, migration)
- Can be overridden by unprivileged userland actions
- Per-task, Per-VMA. Way too much surface area.

*except certain contexts like interrupts

Memory Arenas: N_MEMORY, N_SPM

“It would be nice if the page allocator could operate on different Arenas.”
- $KERNEL_DEV said this to me off hand in a meeting one day

I demonstrated this in my RFC for this talk: Specific Purpose Memory Nodes
- cpuset extension
- memory-tier/hotplug provide exclusive nodelists (proper solution: N_SPM)
- GFP flag

Similar mechanism had previously been proposed, Twice
- essentially adds N_SPM

Opinion: I think between these two proposals, we have the full puzzle.

How it works

1) N_SPM and N_MEMORY do not intersect
   a) All current users of page_alloc default to using N_MEMORY
   b) page_allocator hard-filters nodemask on N_MEMORY
2) cpusets.mems_allowed operates on UNION(N_SPM, N_MEMORY)
   a) a new (non-user facing, internal) cpusets.sysram_nodes provides N_MEMORY filter equivalent
3) GFP_SPM_NODE flag provides a switch to change the filter function
   a) when flag is present, filter on N_SPM instead of N_MEMORY
   b) when flag is present, filter on mems_allowed instead of sysram_nodes

This essentially creates “Private Memory Nodes”

Names are hard: Specific Purpose vs Private - same thing?

What then?

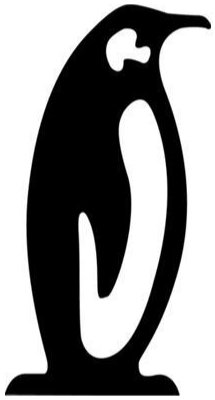
- Maybe pivot ZONE_DEVICE to but used for this.
  - allow things like page_is_spmem() similar to devmem page checks
  - allows clean callback into driver to push complexity out of core kernel

- enlightened drivers that register SPM nodes and can alloc, free, and manage access constraints (Page Table perms, maybe re-use hmm code here).

- **do not** extend mempolicy or other user-facing components to be SPM aware
  - forces complexity into drivers.
  - limited userland exposure means lower long-term risk

FAQ

- What not just use mempolicy?
  - insufficient isolation and explicitly ignored by a variety of users.
- Can’t we just manage with existing cgroups extensions?
  - not all systems use cgroups or cpusets
  - root can’t remove the node (nothing will be able to access at all)
- Do we really need the GFP flag?
  - yes… no… maybe.
  - Provides a much harder isolation guarantee and fewer tripping hazards.
- Why not ZONE DEVICE?
  - Still have to re-implement core services like reclaim (if wanted)
  - Can imagine SPM_NODE using ZONE_DEVICE, so think of this as another ZONE_DEVICE mode?



東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025