Moving swapping infrastructure to Rust

Vitaly Wool, Konsulko AB

Linux Plumbers Conference
Tokyo 2025

About me: Vitaly

Konsulko Group

- Has been working with embedded Linux since 2003
  - MontaVista Europe
- Moved to Sweden in 2009
  - continued working with embedded Linux
  - consulting for Sony, Google, Axis, etc.
- Currently living in Malmö
- Working within the Konsulko Group
- Principal engineer at Konsulko AB

Why *rustify*?

Konsulko Group

- Bigger memory safety
  - Rust is not a silver bullet but still it has an edge over C there
  - NonNull, protection agains buffer overflows, etc.
- More secure
  - Reduced undefined behavior => smaller attack surface
- More robust
  - Emphasis on immutability
- More maintainable/sustainable
  - Powerful error handling mechanisms (e.g. Result, Option)
  - Strict coding style requiring comments

Kernel swapping structure

Konsulko Group

- Generic swap structure
  - Core swap implementation (swap.c, swapfile.c, etc.)
- RAM based compressed swap storages
  - Frontend (zswap, ZRAM)
  - Allocation backend (zsmalloc)
- Shmem (to a certain extent)
  - GPU drivers use it for GPU page swapping

Kernel swapping structure: what can be *rustified*?

Konsulko Group

- Not all architectures support Rust for Linux [yet]
  - Core components should remain C based
  - zswap should remain C based
  - Shmem should remain C based
- zsmalloc could be a good candidate
  - More considerations will follow
- ZRAM is not a core component
  - Block device driver, basically a ramdisk on steroids
  - Used almost exclusively by x86_64 and aarch64

zsmalloc

Konsulko Group

- Seasoned allocator backend
  - Provides very good compression density
- Rather complex implementation
  - Objects are divided into 255 size classes
  - objects of the same class stored consequently within a page
  - Some objects span across 2 pages
- Doesn’t work well in 16+K page setups
  - Lower granularity
- Currently the only available allocator backend
  - Can’t be moved to Rust due to the architecture limitations

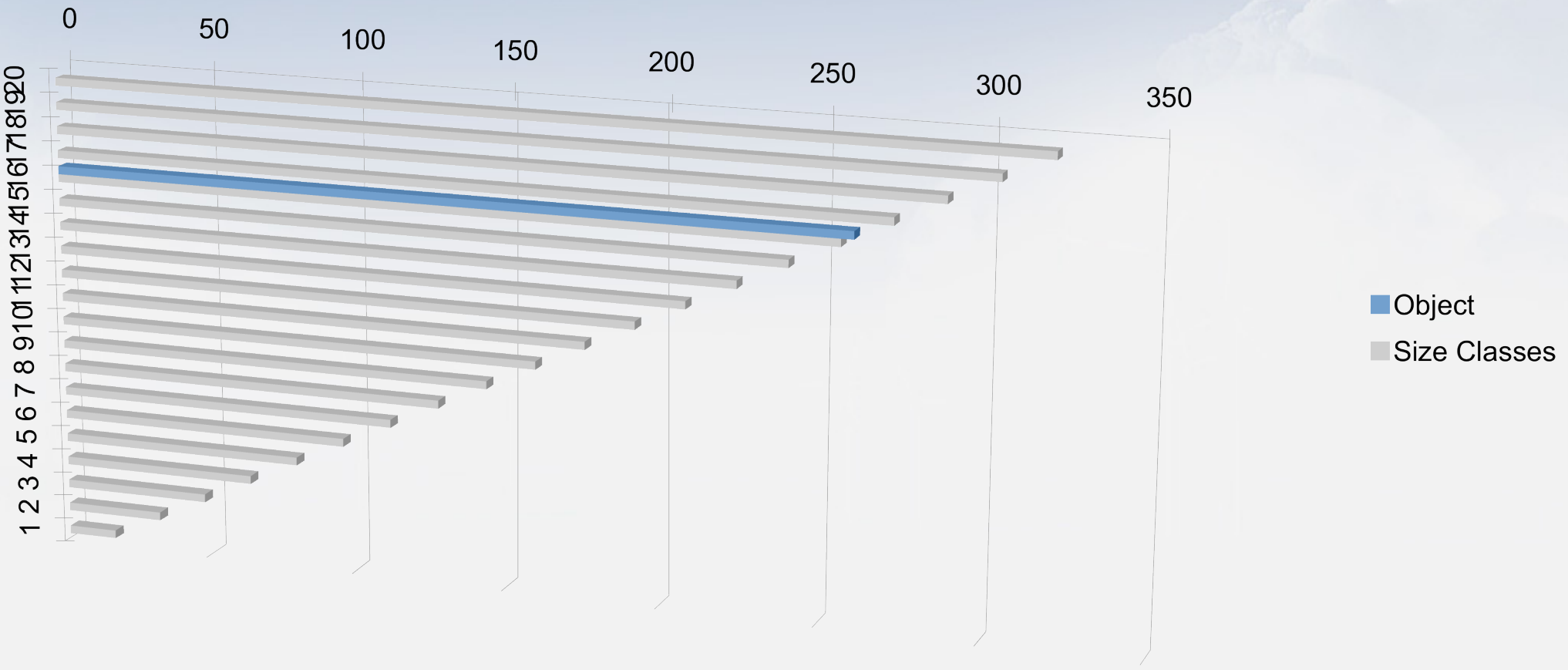
zblock

Konsulko Group

- If we can't move zsmalloc to Rust, we can implement an alternative allocator from scratch
- zblock: based on 2 simple ideas
  - Use page blocks instead of single pages
  - Divide large blocks into an array of same size objects
  - These same size objects (slots) don't have to be of 2^x size
- Small code footprint and easy to understand concept
- How it relates to zsmalloc
  - 4K pages: comparable compression density, better performance
  - 16K pages: better compression density, better performance

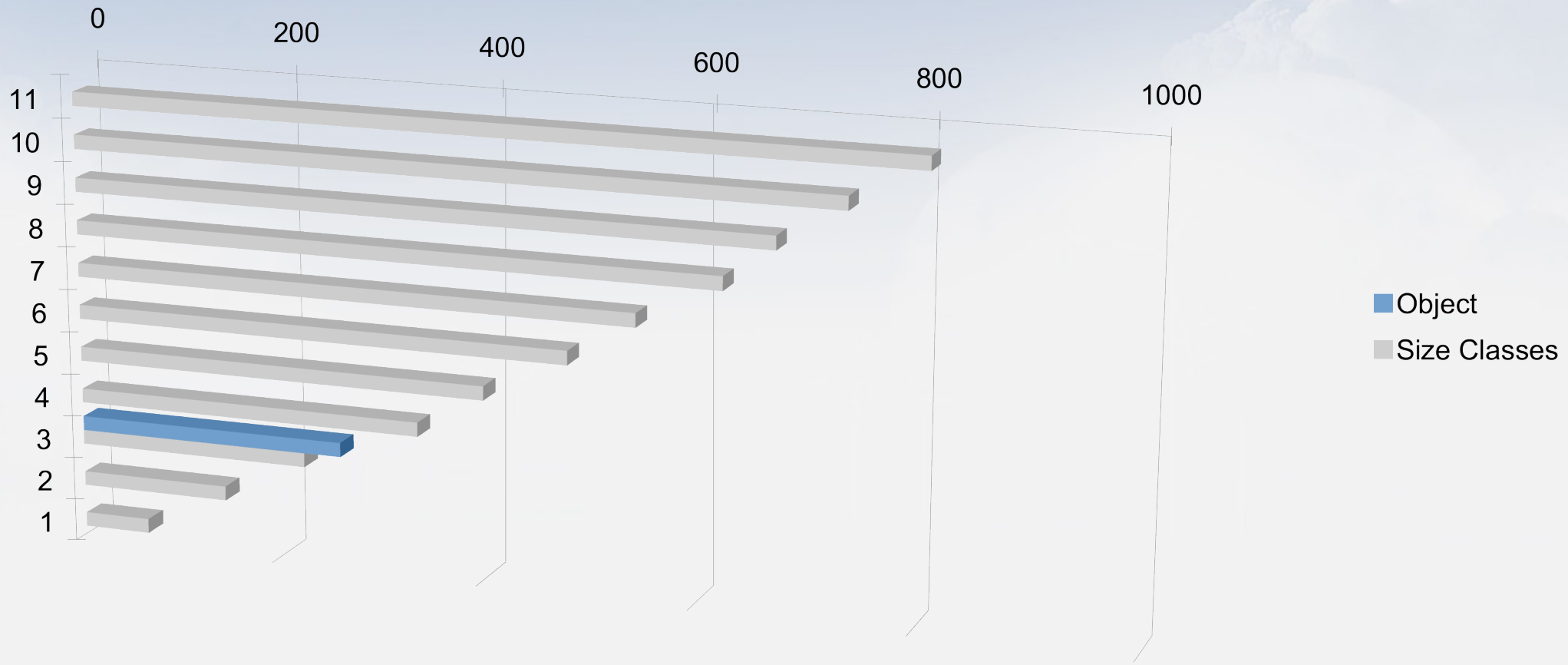
zsmalloc size classes

Konsulko Group



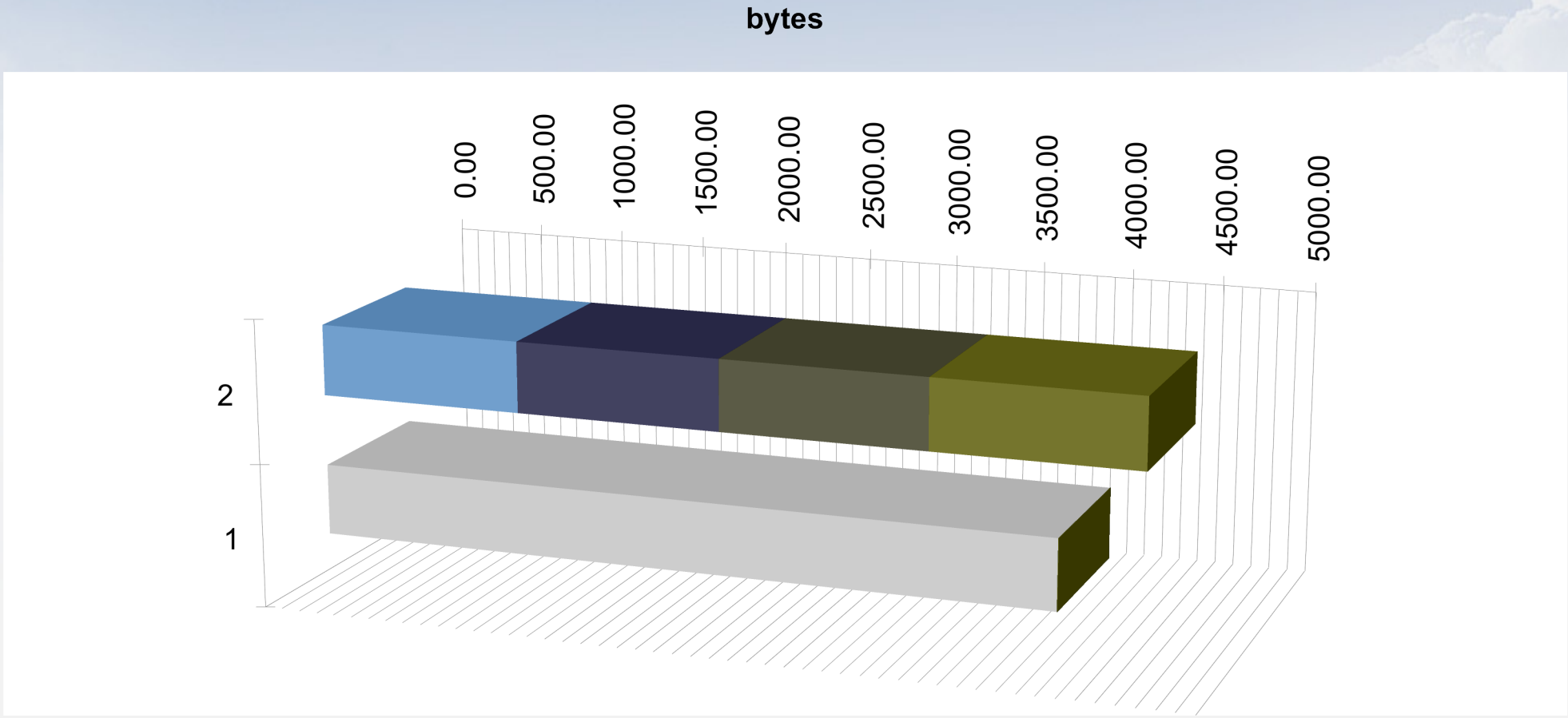
zblock size classes

Konsulko Group



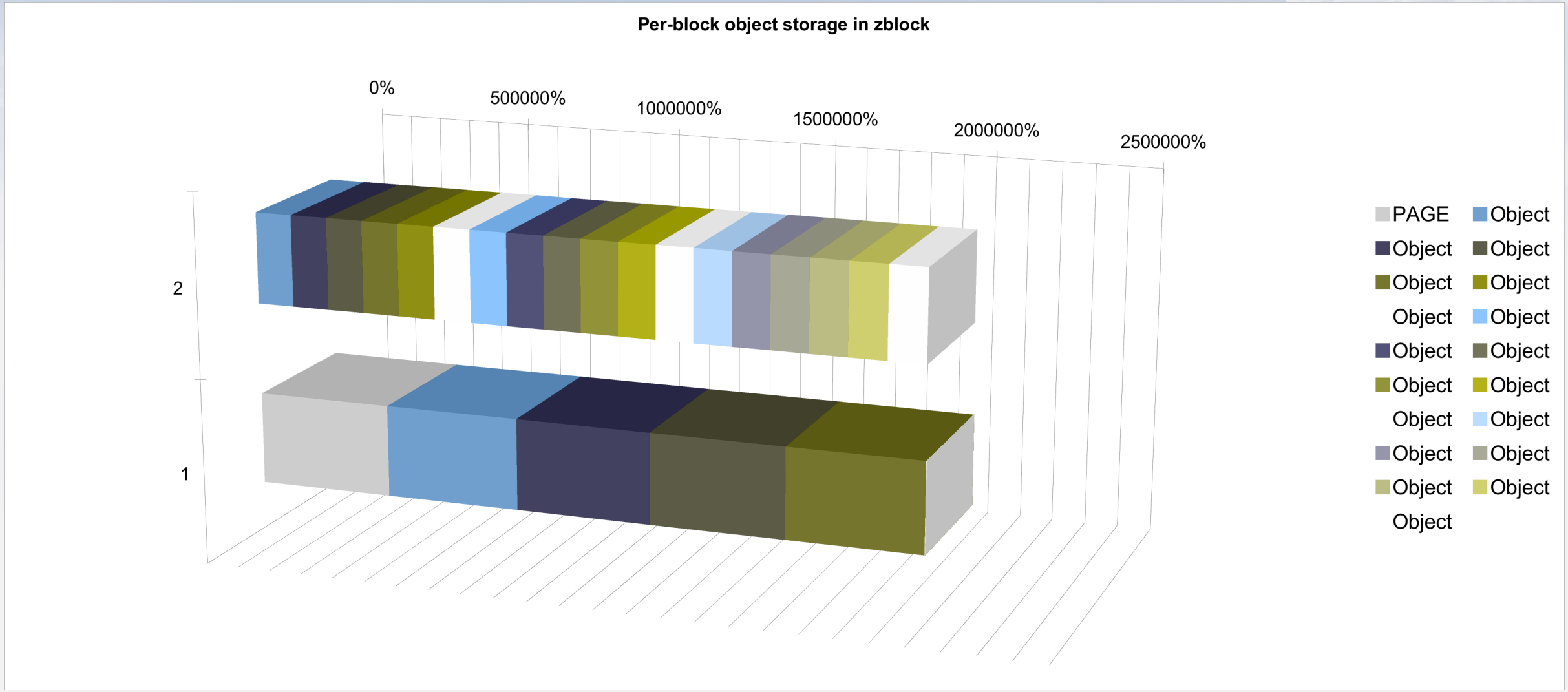
zsmalloc's spillover

Konsulko Group



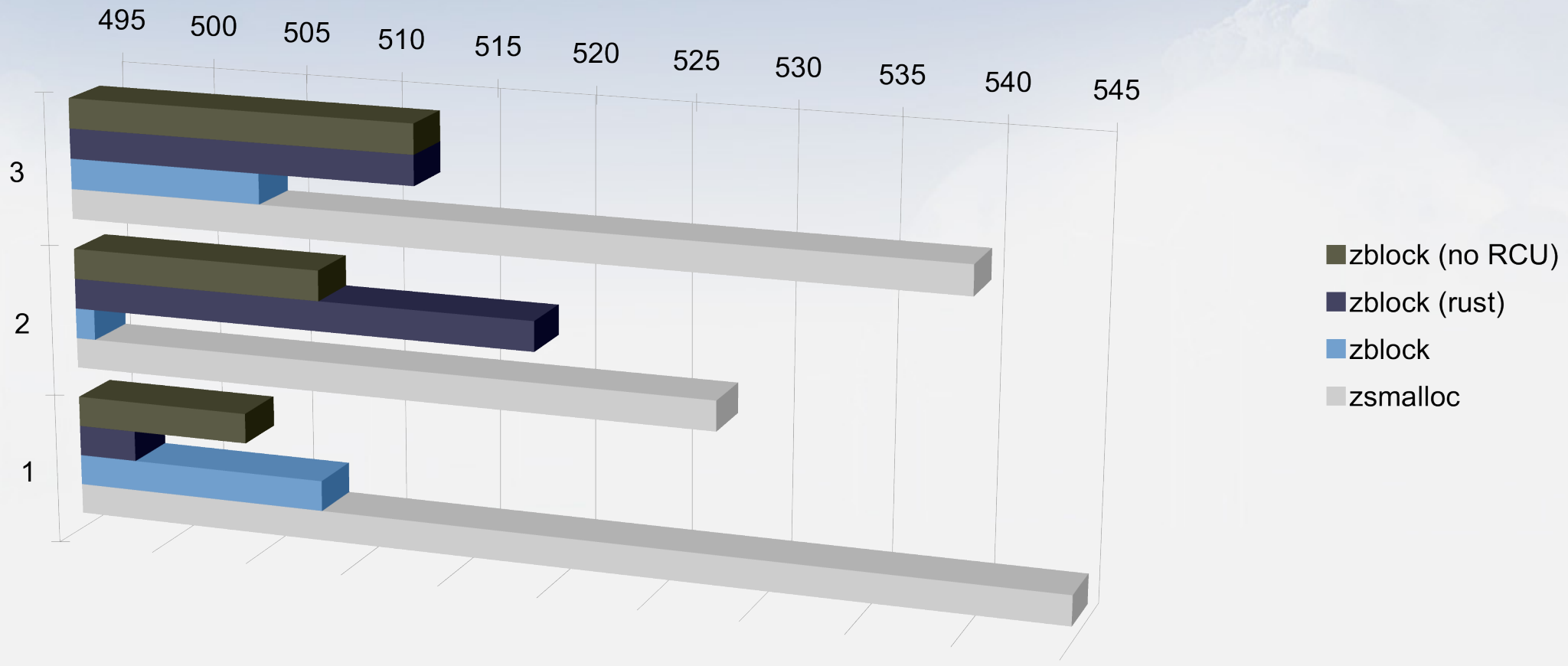
zblock: no spillover

Konsulko Group



Quick performance comparison

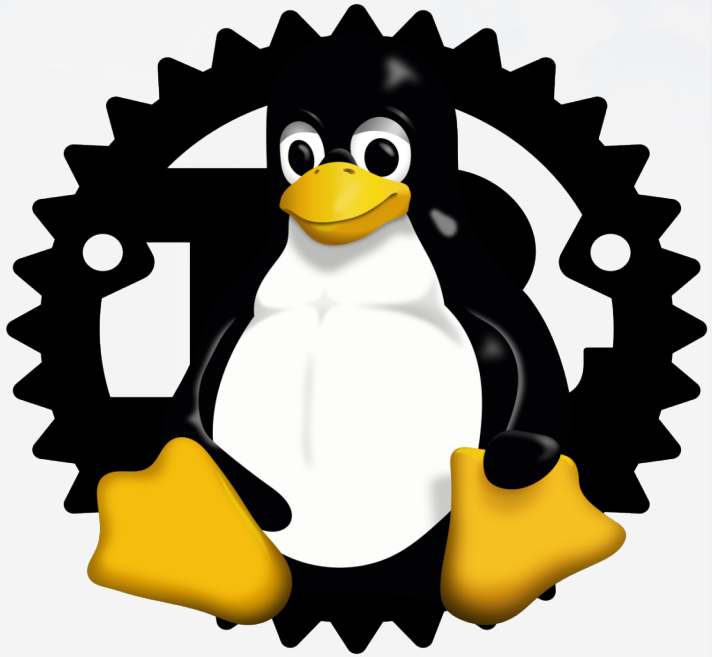
Konsulko Group



Rust allocator: summary

Konsulko Group

- Not for all architectures
- New algorithm (page block based)
- Fast and scalable
- Higher memory safety
- Native Rust API
- Implemented, submission under way



Rust compressed ramdisk

Konsulko Group

- Why
  - ZRAM is bloated
  - Blocks of unused code
- How
  - Zblock.rs provides a safe native Rust API
  - NeoZRAM uses that API
- Status
  - Work ongoing

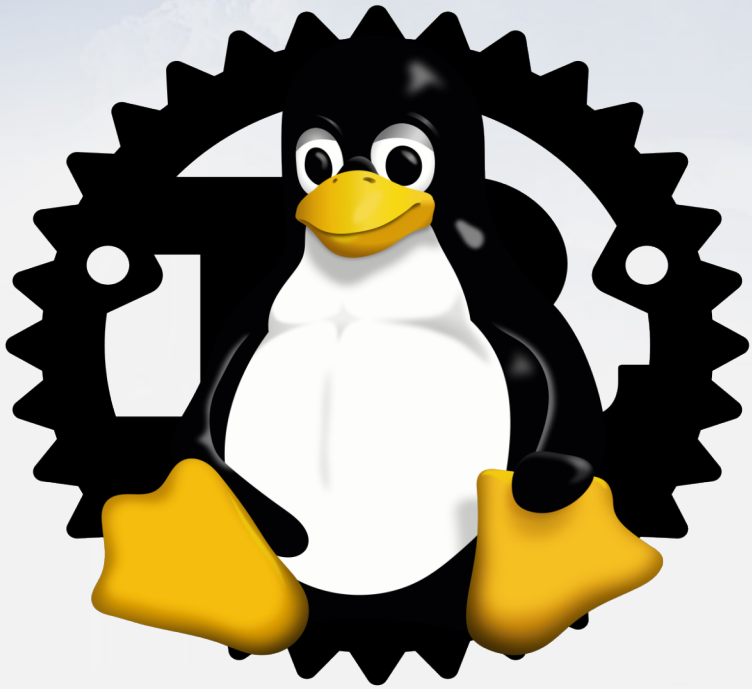
Shmem and Rust?

Konsulko Group

- Shmem can’t be moved to Rust (should work on all architectures)
- Rust DRM drivers could benefit
- Using shmem for DRM buffer objects is suboptimal
  - BOs often span across very many pages
  - BO pages are treated independently
  - However, BOs are swapped in/out in their entirety
- Idea: implement streaming compression for large BOs
  - No need to use special allocators like zsmalloc
  - Some preliminary work has started

Thanks for your attention!

E-mail: vitaly.wool@konsulko.com



LINUX PLUMBERS CONFERENCE 2025
TOKYO, JAPAN