Swapping and embedded:
compression is the key

Vitaly Wool

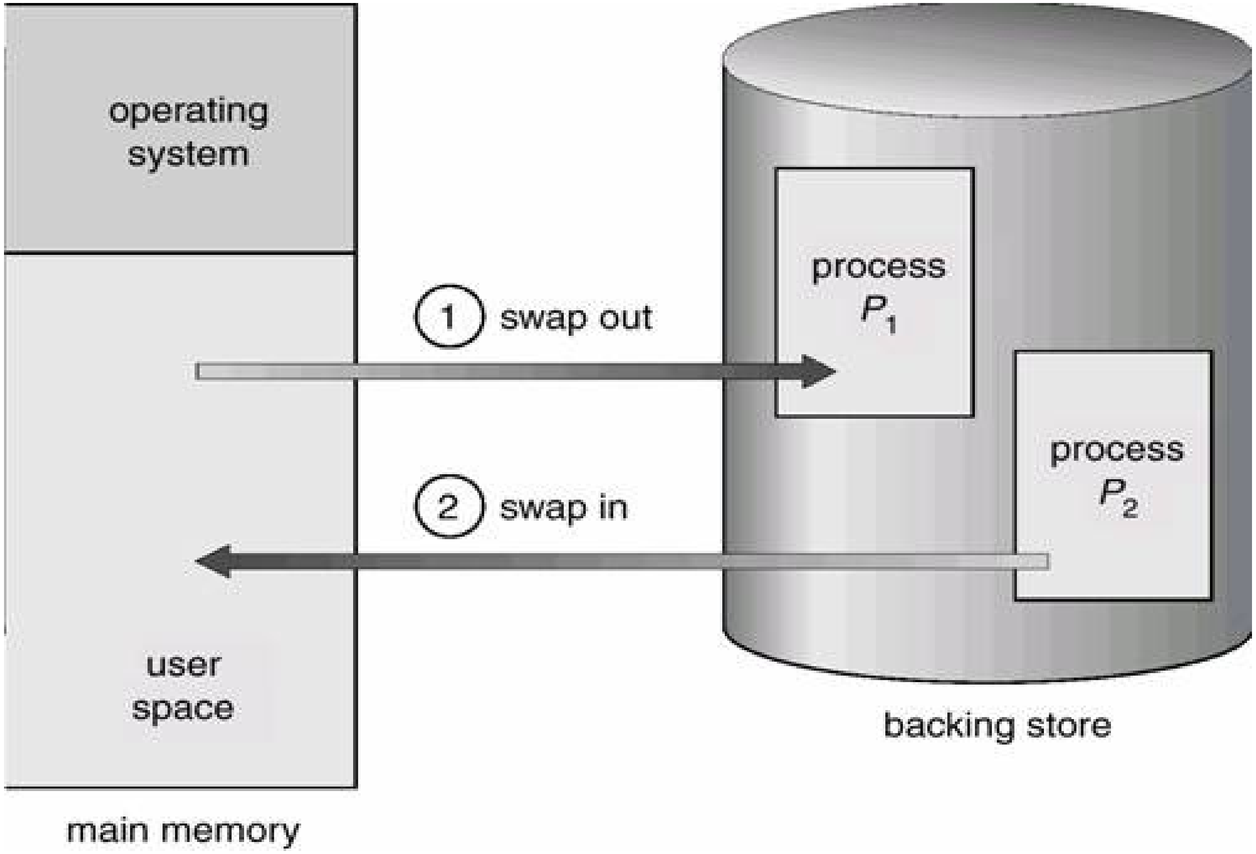
Embedded Linux Conference Europe 2016

Swapping (Paging)

- Paging: [OS capability of] using a secondary storage to store and retrieve data
  - With RAM being primary
  - Storing and retrieving happens on a per-page basis
- Page
  - Uni-size storage block, usually of size 2^n
  - Corresponds to a single record in page table
- Paging is only possible with VM enabled

Intro>

Swapping



Embedded device objectives

- [very] limited RAM
- [relatively] slow storage
  - Using swap will hurt performance
- [relatively] small storage
  - Hardly is there a place for big swap
- Flash chip used as a storage
  - Swap on flash wears it out fast

Intro>

Swapping in Embedded

- Should be applicable
  - Constrained RAM
- But is isn't sometimes
  - Constrained storage
- May have adverse effects
  - Flash storage faster wear-out
  - Longer delays if the storage device is slow
- There has to be a way out...

Swapping optimization: zswap

- zswap: compressed write-back cache for swapped pages
  - Write operation completion signaled on write-to-cache completion
- Compresses swapped-out pages and moves them into a pool
  - This pool is dynamically allocated in RAM
- Configurable parameters
  - Pool size
  - Compression algorithm

Smarter swapping>

zswap backend: zbud

- zbud: special purpose memory allocator
  - allocation is always per-page
- Stores up to 2 compressed pages per page
  - One bound to the beginning, one to the end
  - The in-page pages are called “buddies”
- Key characteristics
  - Simplicity and stability
- zbud is *the* allocator backend for zswap

Smarter swapping>

RAM as a swap storage

- Compression required
  - No gain otherwise
  - But increases CPU load
- Implementation of a [virtual] block device required
- Careful memory management is required
  - Should not use high-order page allocations

Smarter swapping>

ZRAM

- Block device for compressed data storage in RAM
  - Compression algorithm is configurable
  - Default algorithm is LZO
  - LZ4 is used mostly
- Usually deployed as a self-contained swap device
  - The size is specified in runtime (via sysfs)
  - Configuration is the same otherwise

ZRAM vs Flash swap

- Compared on Carambola (MIPS24kc)
  - Details on the configuration will follow
- Standard I/O measurement tools
  - 'fio' with 'tiobench' script
- Results
  - Average read speed: 730 vs 699 (kb/s)
  - Average write speed: 180.5 vs 172 (kb/s)
- Difference is larger where RAM is faster

Smarter swapping>

zsmalloc: ZRAM backend

- Special purpose pool-based memory allocator
- Packs objects into a set of non-contiguous pages
  - ZRAM calls into zsmalloc to allocate space for compressed data
  - Compressed data is stored in scattered pages within the pool

zsmalloc and zbud compared

	zsmalloc	zbud
Compression ratio	High (3x – 4x)	Medium/Low (1.8x – 2x)
CPU utilization	Medium/High	Medium
Internal fragmentation	yes	no
Latencies	Medium/Low	Low

zpool: a unified API

- Common API for compressed memory storage
- Any memory allocator can implement zpool API
  - And register in zpool
- 2 main zpool users
  - zbud
  - zsmalloc

z--- in detail>

zswap uses zpool API!

- zswap is now backend-independent
  - As long as the backend implements zpool API
- zswap can use zsmalloc
  - Better compression ratio
  - Less disk/flash utilization

ZRAM moving forward>

What if ZRAM used zbud?

- Persistent storage is not used anyway
  - Compression ratio may not be the key
- No performance degrade over time
- Less dependency on memory subsystem
- CPU utilization may get lower
- Throughput may get higher
- Latencies may get lower

Why can't ZRAM use zbud?

- zbud can't handle PAGE_SIZE allocations
  - Uses small part of the page for internal structure
    - Called **`struct zbud_header`**
  - Easy to fix: it can go to **`struct page`**
- ZRAM doesn't use zpool API
  - zsmalloc API fits zpool API nicely
  - Easy to fix: just implement it

ZRAM moving forward>

Allow ZRAM to use zbud

- An initiative taken by the author
  - Allow PAGE_SIZE allocations in ZBUD
  - Make ZRAM use zpool
- Two mainlining attempts
    - https://lkml.org/lkml/2015/9/14/356 [1]
    - https://lkml.org/lkml/2015/9/22/220 [2]
  - Faced strong opposition from ZRAM authors
  - Vendor neutrality questionable
- More attempts to come

New better allocator?

- Requirements
  - Higher compression ratio than that of zbud
  - More determinism than in zsmalloc
  - Less fragmentation issues than in zsmalloc
- Idea
  - Do like zbud, but up to 3 objects per page
- Implementation
  - Z3fold: in mainline since 4.7

z3fold: new allocator>

Key features

- Up to 3 objects per page
  - Compression ratio up to 3x
  -
  -
- Objects can not cross page boundary
  - More determinism than in zsmalloc
  -
- Implementation
  - Z3fold: in mainline since 4.7

Prerequisites

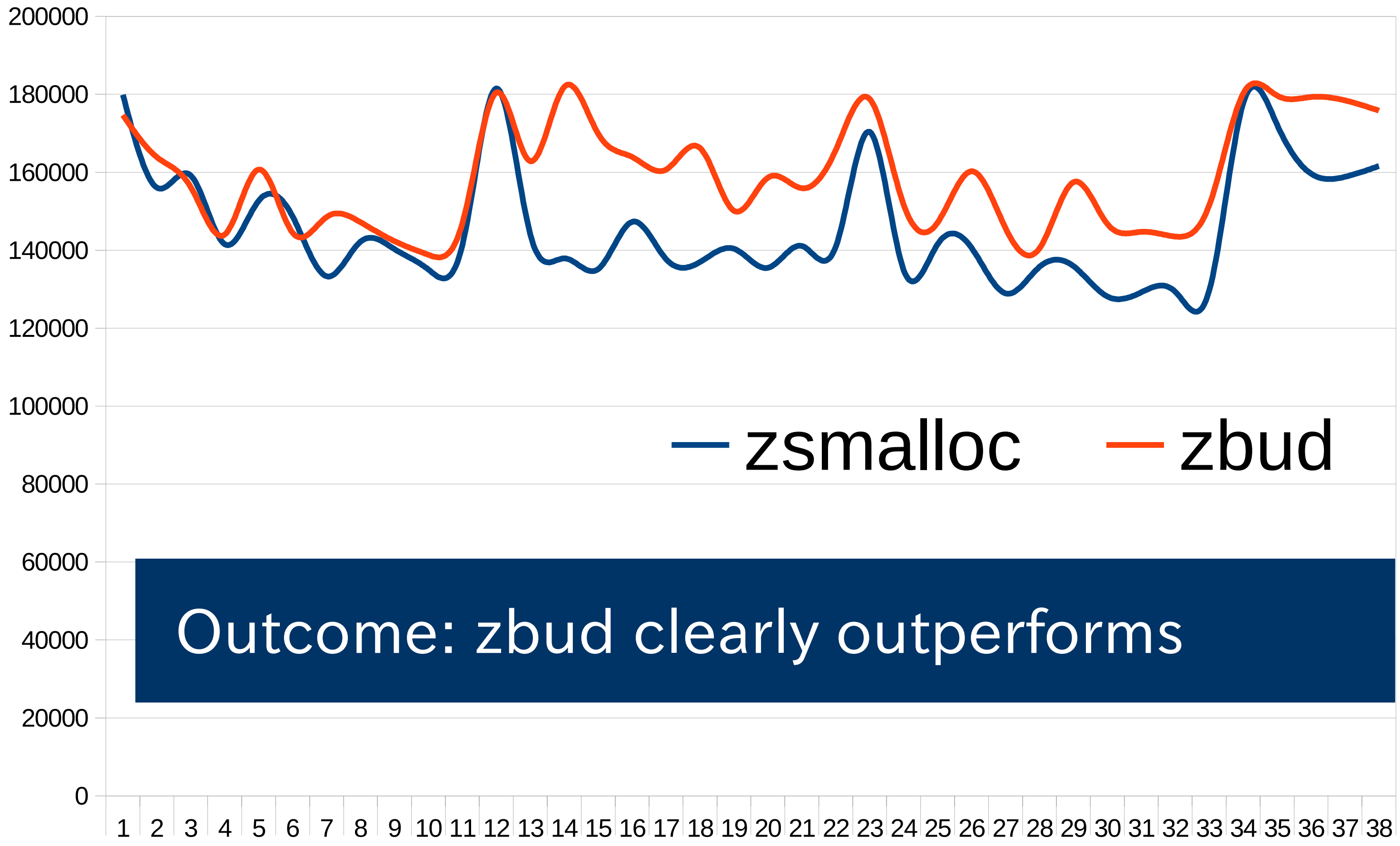
- Use fio for performance measurement
  - Written by Jens Axboe
  - Flexible and versatile
- EXT4 file system on /dev/zram0
  - 50% full
- A flavor of fio 'enospc' script
  - Adapted for smaller block device (zram)
- 40 iterations per z--- backend (zbud/zsmalloc)

Test device 1

- Sony Xperia Z2
  - MSM8974 CPU
    - 2.3 GHz Quad-Core Krait™
  - 3 GB RAM
- Cyanogenmod build as of Jan 15, 2016 (12.1)
  - A flavor of Android 5.1.1
  - Custom 3.10-based kernel

Measurements>

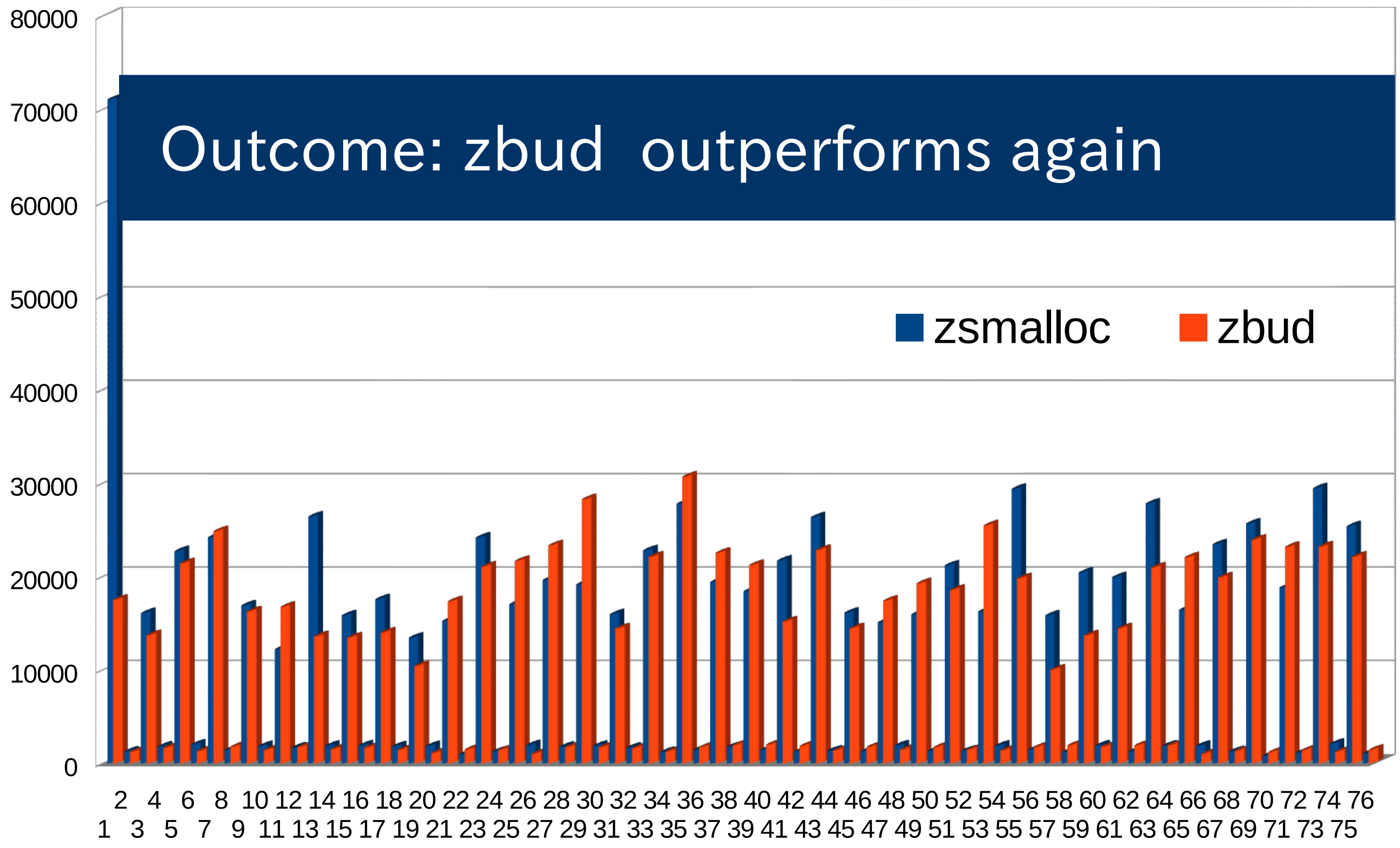
ZRAM performance: Android



Outcome: zbud clearly outperforms

Measurements>

ZRAM latency: Android



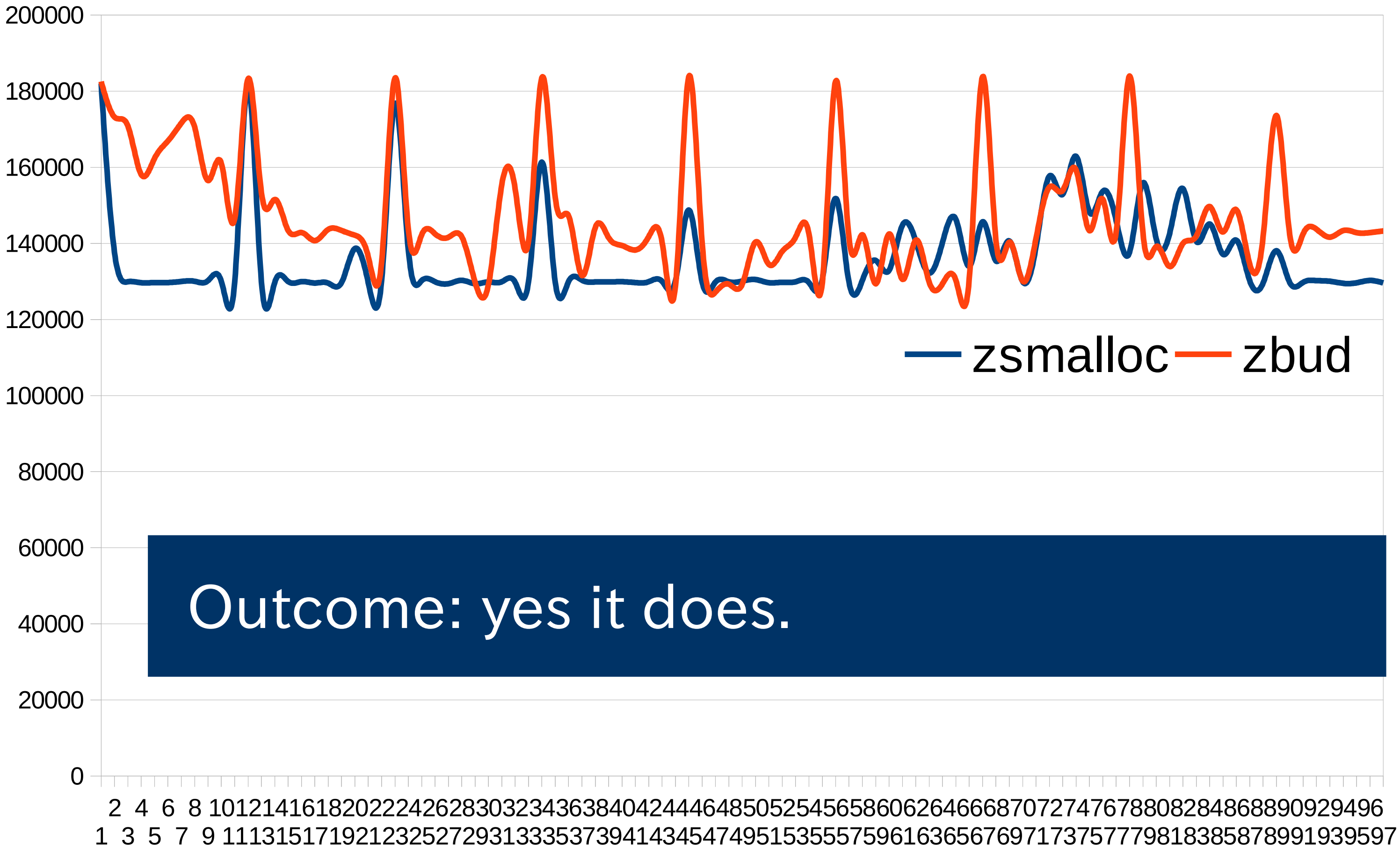
Outcome: zbud outperforms again

ZRAM performance: Android

Okay what happens in the long run, does zbud remain superior to zsmalloc?

Measurements>

ZRAM performance: Android

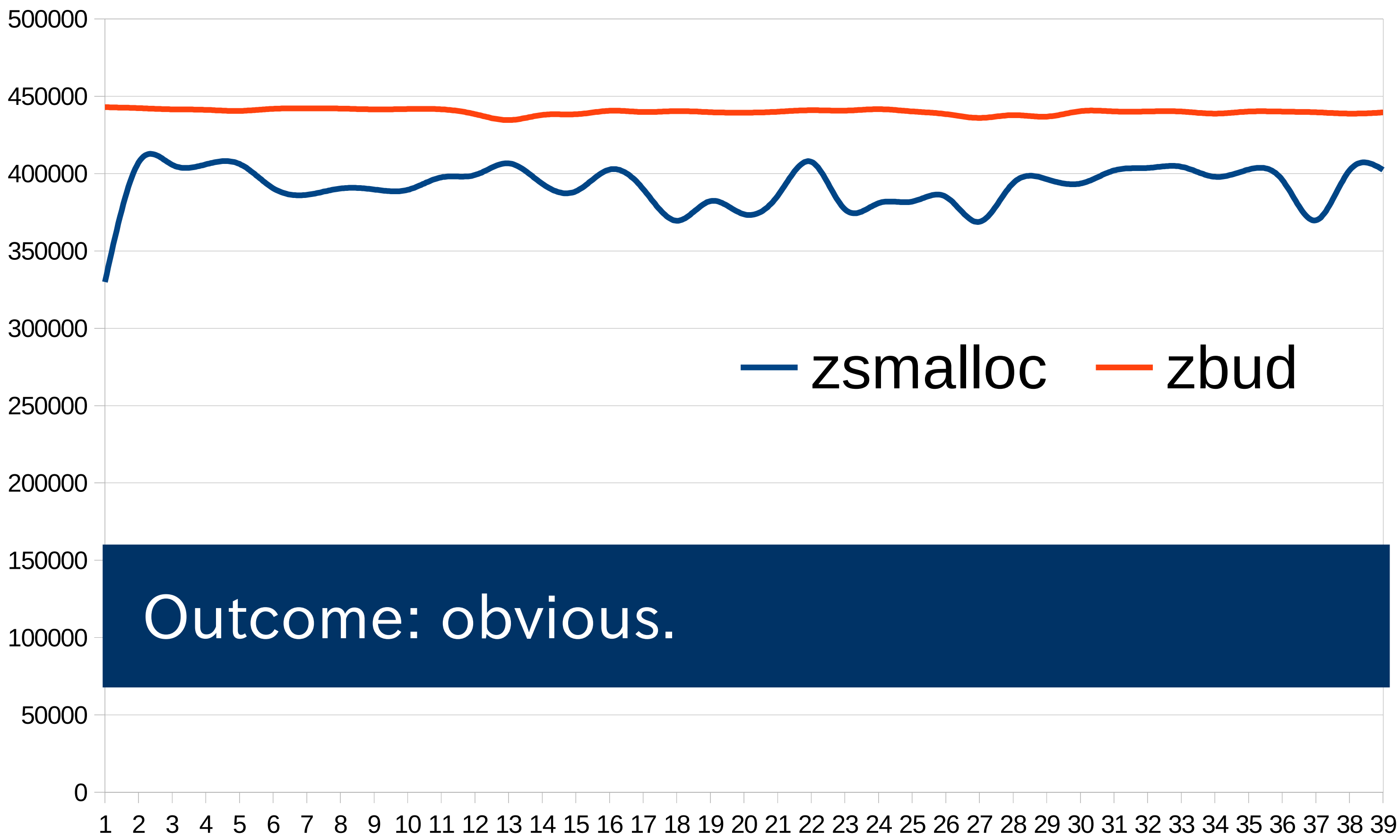


Outcome: yes it does.

Test device 2

- Intel Minnowboard Max EVB
  - 64bit Atom™ CPU E3815 @ 1.46GHz
  - DDR3 2 GB RAM
  - Storage 4 GB eMMC
- Debian 8.4 64 bit
  - Custom 4.3-based kernel

ZRAM performance: x86_64

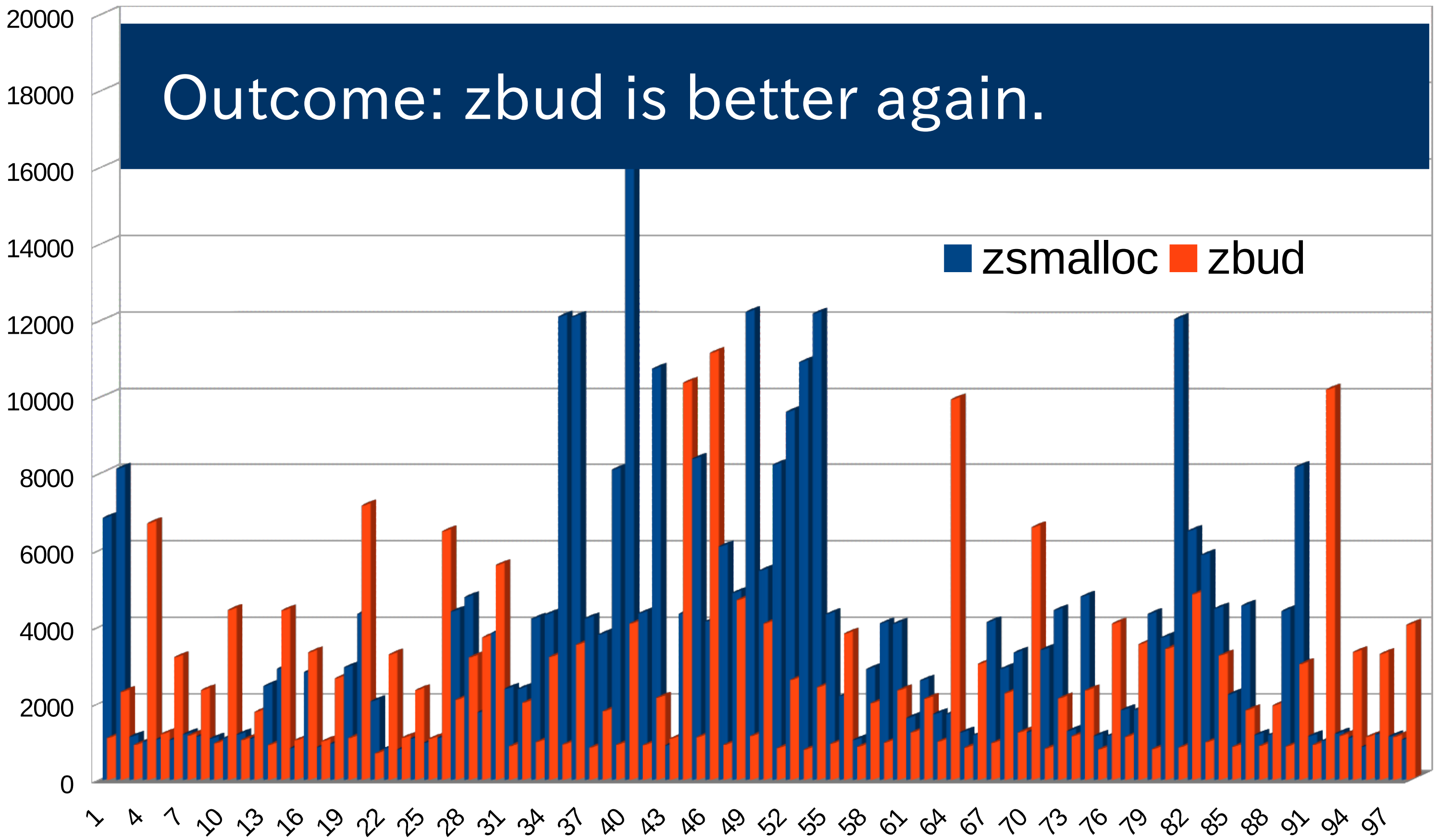


Outcome: obvious.

Measurements>

ZRAM latency: x86_64

Outcome: zbud is better again.



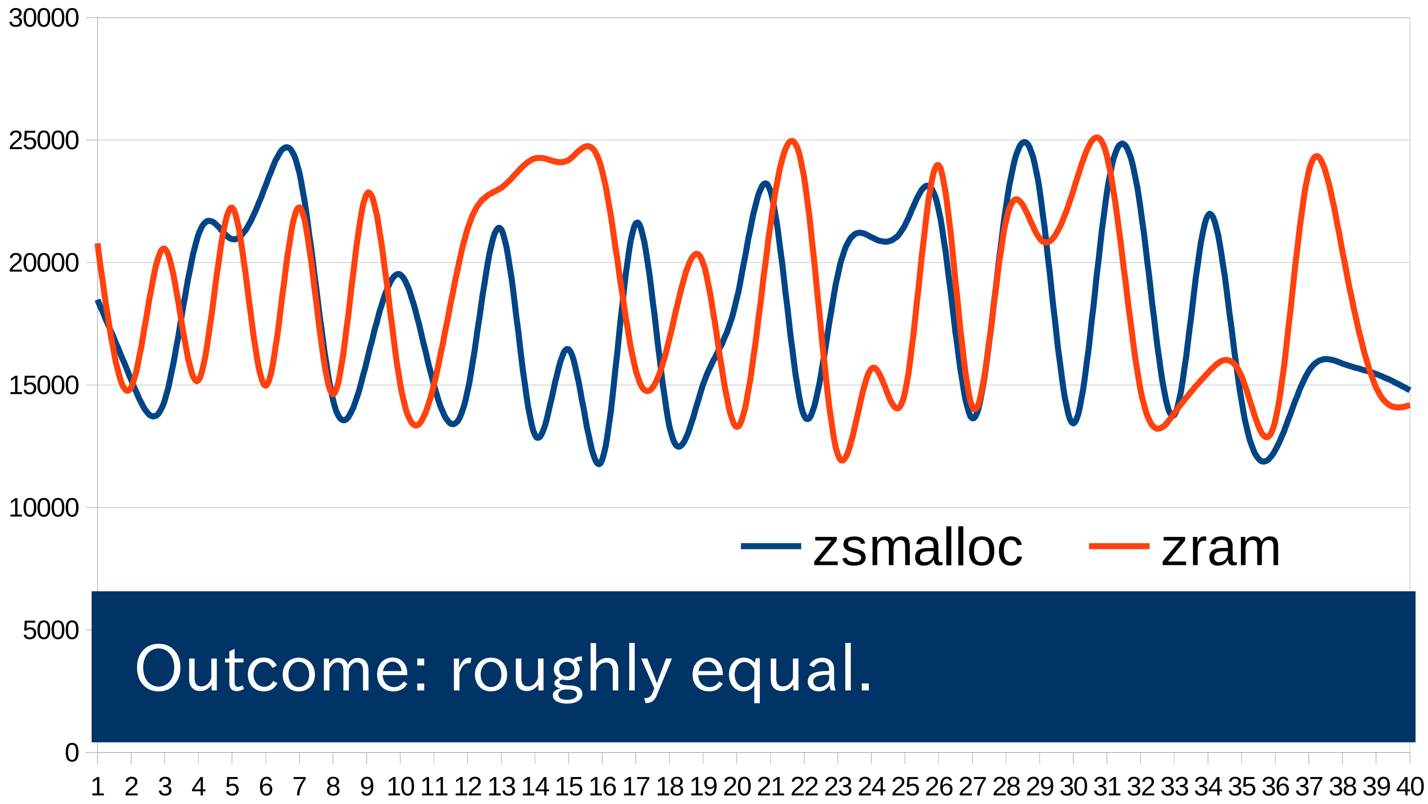
Measurements>

Test device 3

- Carambola 2
  - MIPS32 24Ke
  - Qualcomm/Atheros AR9331 SoC
  - 400 MHz CPU
  - 64 MB DDR2 RAM
  - Storage 512 MB NAND flash
- OpenWRT
  - Git as of Jan 15, 2016
  - Custom 4.3-based kernel

Measurements>

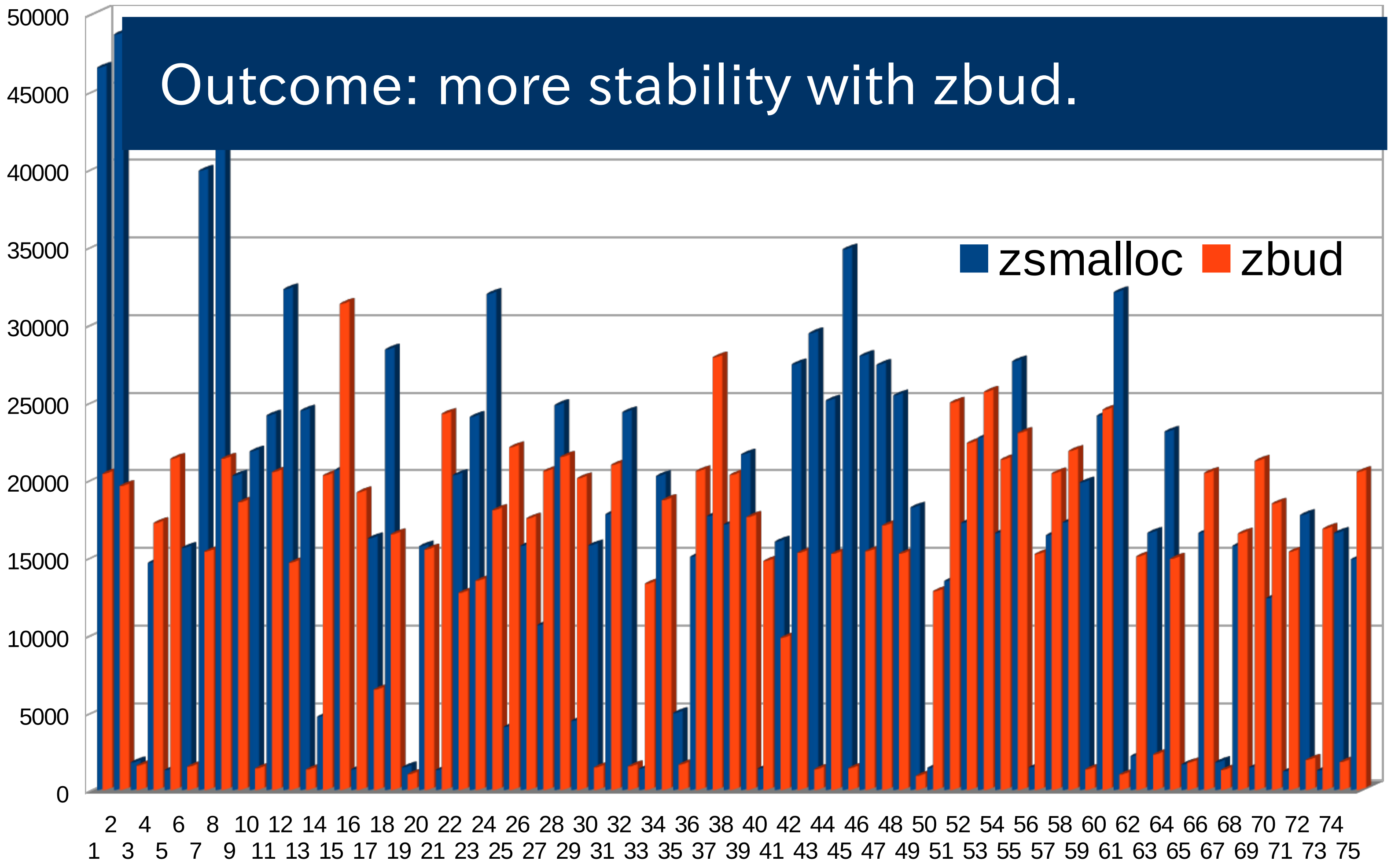
ZRAM performance: MIPS32



Outcome: roughly equal.

Measurements>

ZRAM latency: MIPS32



Wrap-Up

- Compressed RAM swap is a good idea
  - Many systems can benefit from it
- Two implementations mainlined
  - Zswap: mostly targeting big systems
  - ZRAM: mostly for embedded / small systems
- Each has its own backend
  - zsmalloc for ZRAM, zbud for zswap
- New backend: z3fold
  - Evaluation and measurements ongoing

Conclusions

- Compressed RAM swap is *the* way out for embedded systems
- ZRAM is a better fit for embedded than zswap
- ZRAM backend choice should not be only zsmalloc
  - Zbud can fit nicely in special cases
  - Z3fold is a good alternative in most cases

swapping completed.

Questions?

mailto: vitalywool@gmail.com