Linaro connect
Bangkok 2016

AOSP RAM reduction project retrospective

Presented by

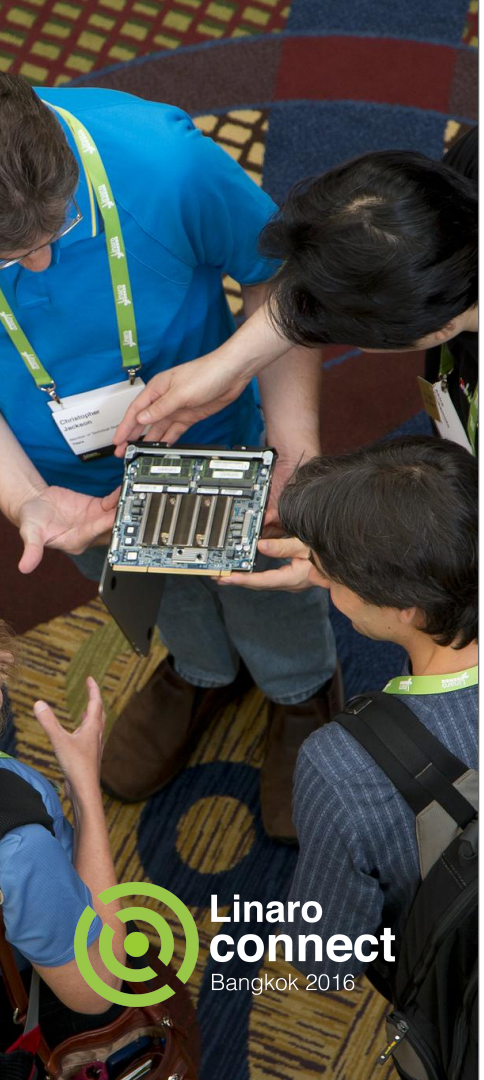
Bernhard “Bero” Rosenkränzer,
YongQin Liu

Date

BKK16-206 March 8, 2016

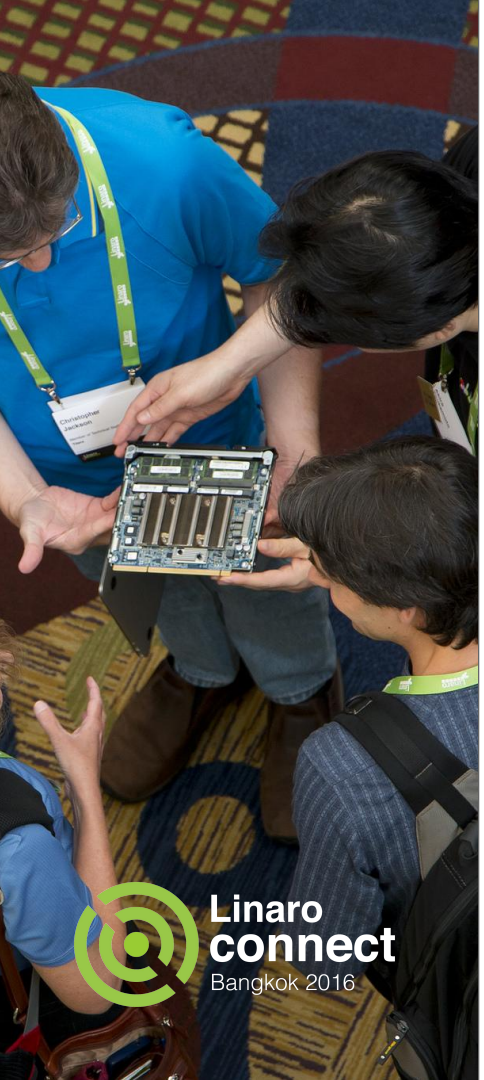
Event

Linaro Connect BKK16



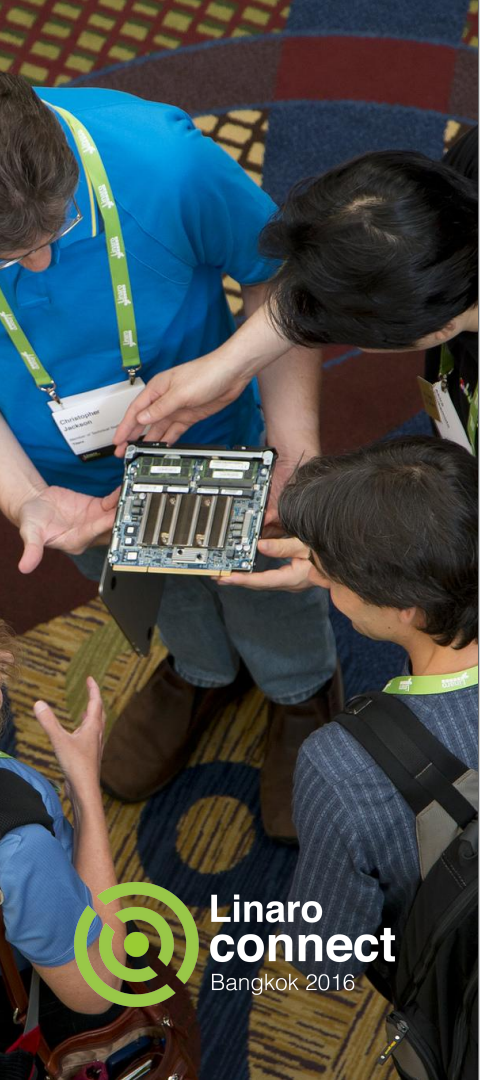
The Goal:

Reduce AOSP memory requirements without hurting performance too badly



What did we try?

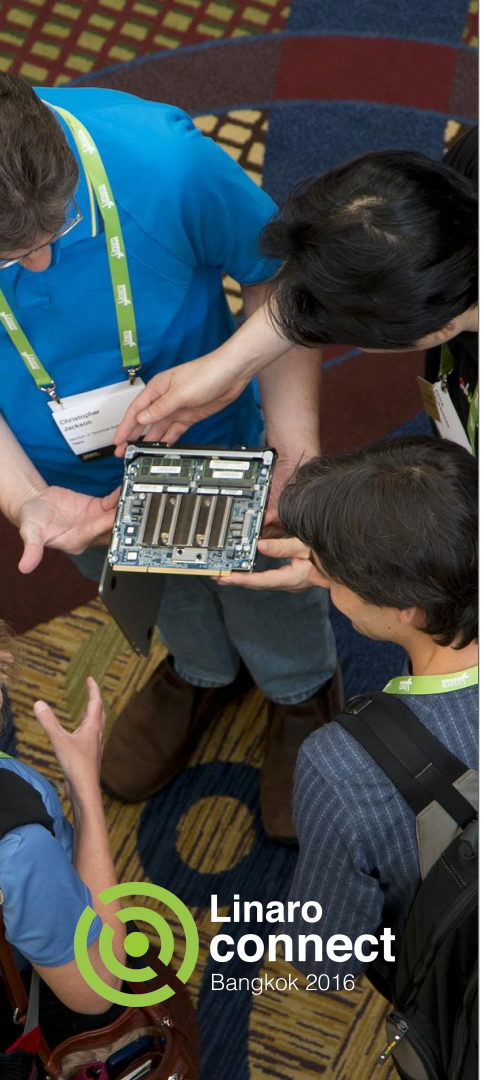
- Update toolchains
- Make use of new features in updated toolchains
- Split libraries into smaller parts
- Tweak settings
- Replace the memory allocator



Update Toolchains

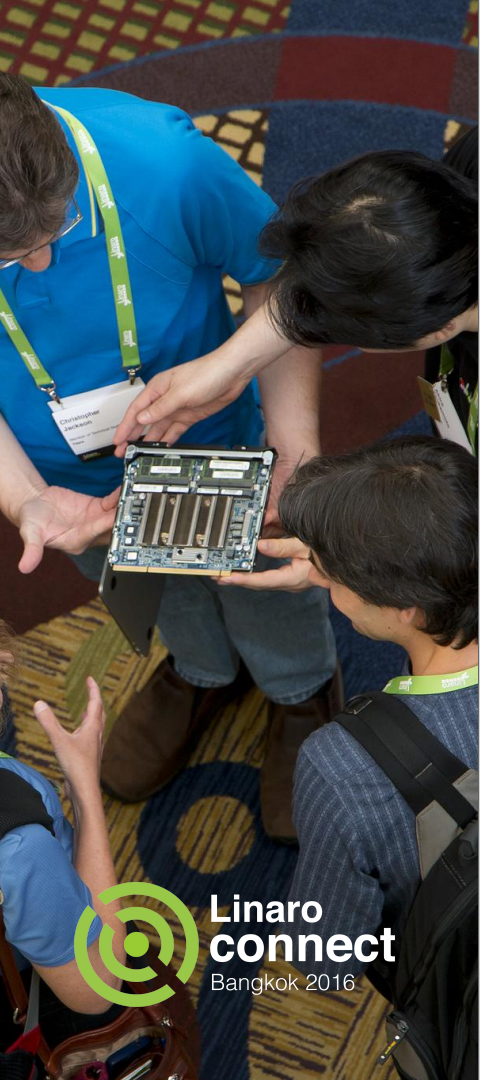
In Android M, gcc 4.9 is used to compile most parts of the OS.

We've replaced it with Linaro gcc 5.3 and updated binutils to 2.26 to go along with it.



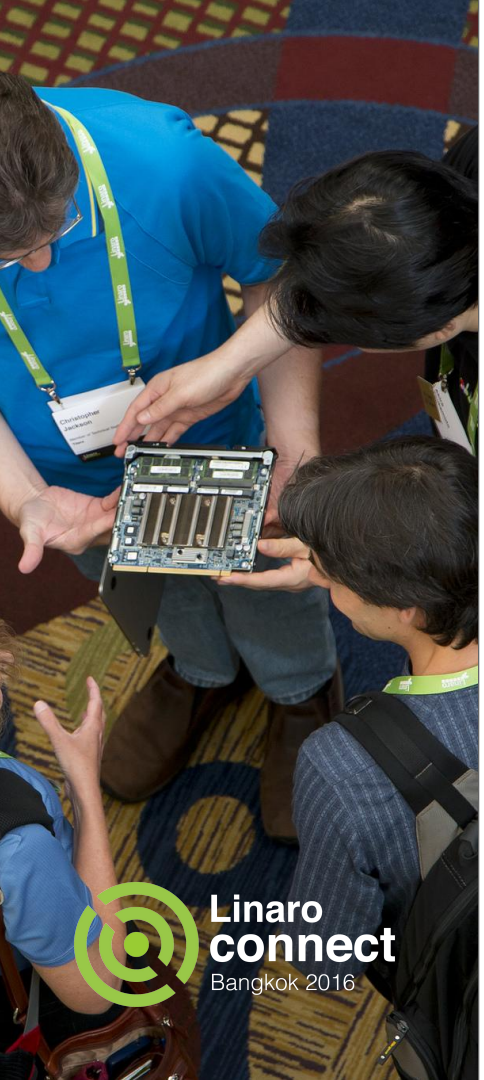
Make use of new toolchain features

New toolchains give us new features that aren't activated by default - e.g. `-mcpu=cortex-a57 -mfpu=neon-fp16` etc. can generate faster and smaller code (for maximum effect, needs `-ffast-math` so regular FP instructions can be offloaded to the NEON unit)



Make use of new toolchain features

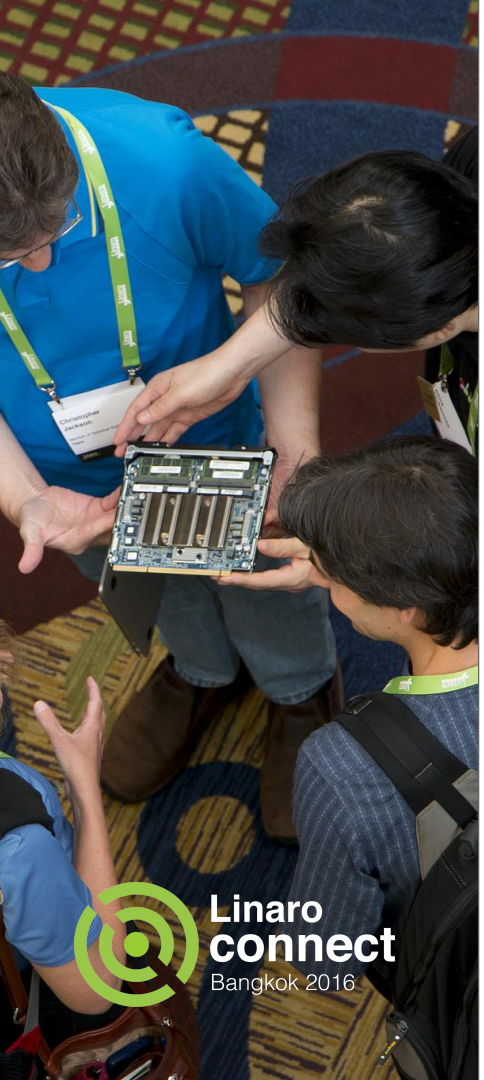
Some helpful toolchain features exist even in 4.9, but aren’t used…



Make use of new toolchain features

`-Bsymbolic` - binds references to global symbols to the definition within the shared library.

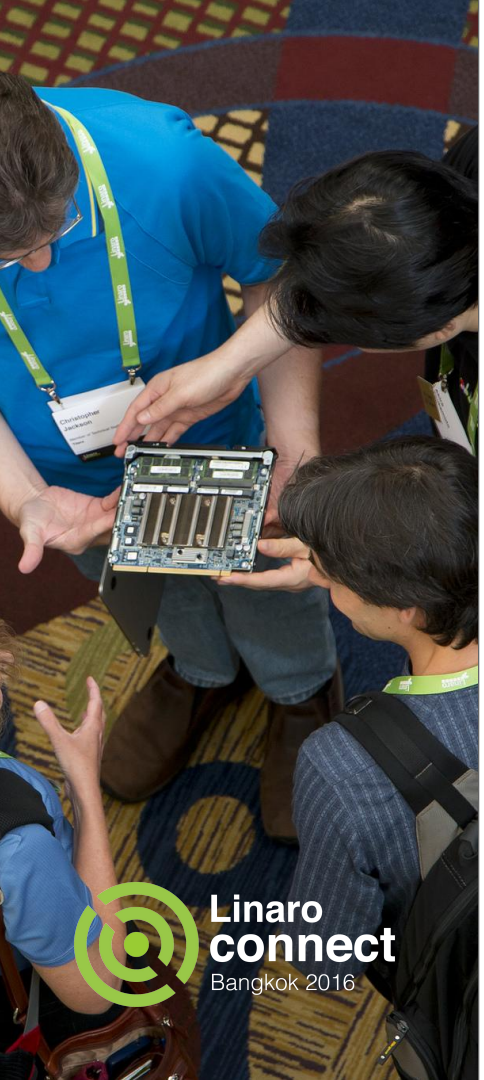
Generates smaller and more memory efficient code, at the cost of breaking `LD_PRELOAD` etc. (which isn't used in regular AOSP use anyway)



Make use of new toolchain features

`-fvisibility=hidden/`
`-fvisibility=protected`

Hide symbols to the outside world - allows throwing some code away



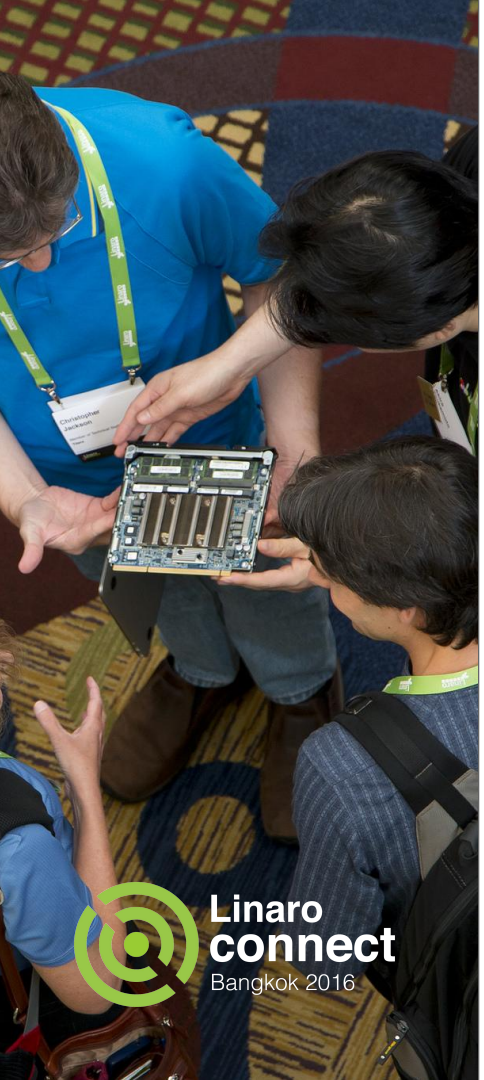
Split libraries into smaller parts

Many libraries combine very commonly used functionality and rarely used functionality, e.g.

libpng/libjpeg/libgif:

Decoder: Used by launcher (icons), gallery, browser, …

Encoder: Rarely used

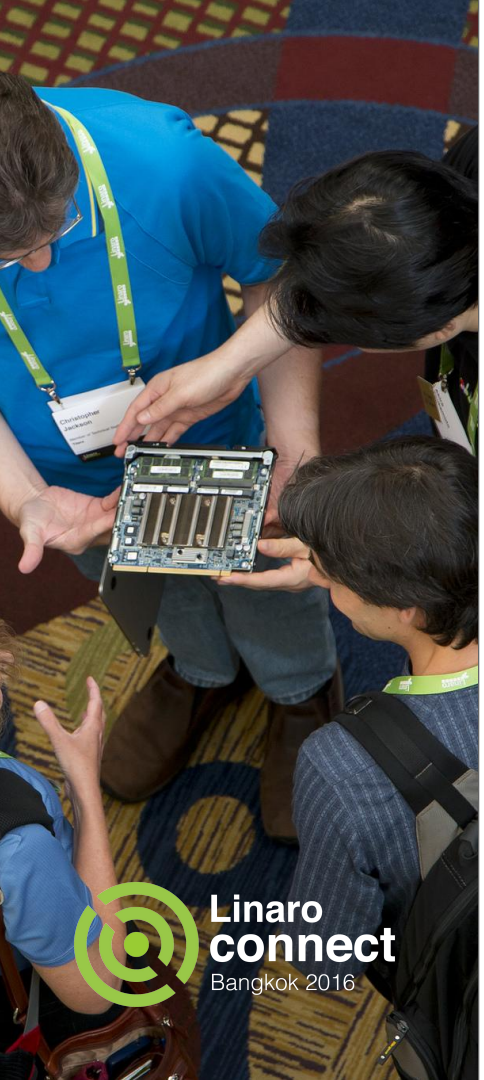


Split libraries into smaller parts

zlib:

Decompression: Used frequently (esp. as part of libpng)

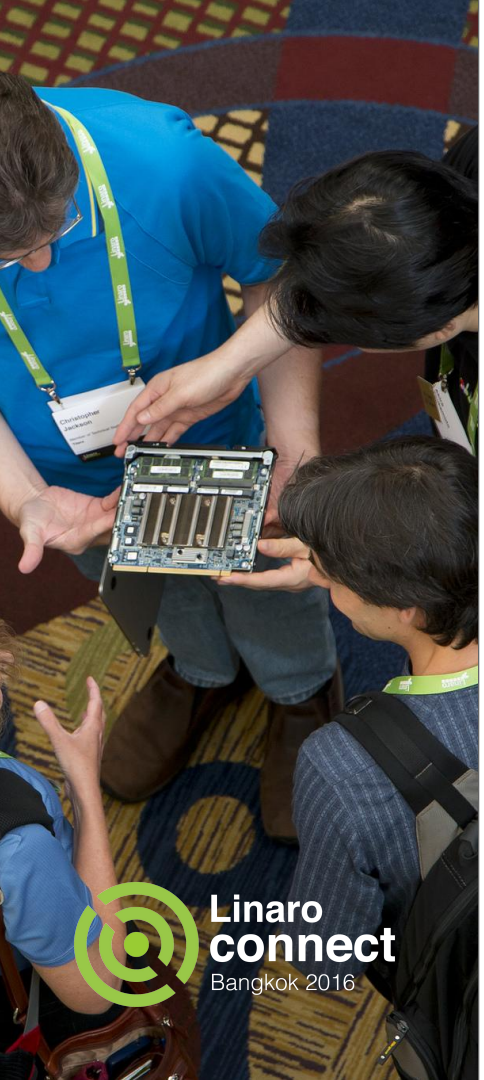
Compression: rarely used



Tweak settings

ART has numerous settings that affect its memory use - e.g.:

```
dalvik.vm.heapstartsize=8m
dalvik.vm.heapgrowthlimit=64m
dalvik.vm.heapsize=174m
dalvik.vm.heaptargetutilization=0.75
dalvik.vm.heapminfree=512k
dalvik.vm.heapmaxfree=2m
```

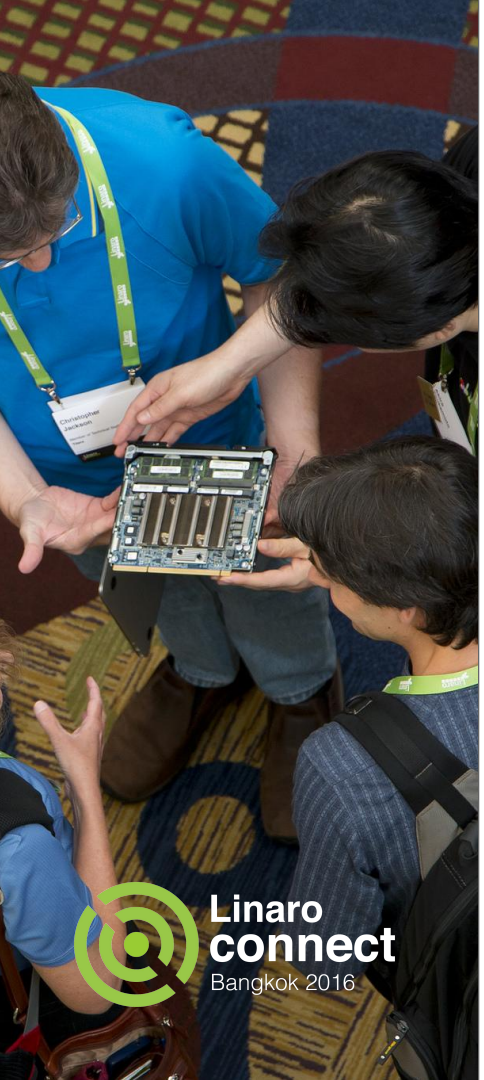


Replace the memory allocator

Bionic comes with two allocators - jemalloc and dlmalloc.

jemalloc:

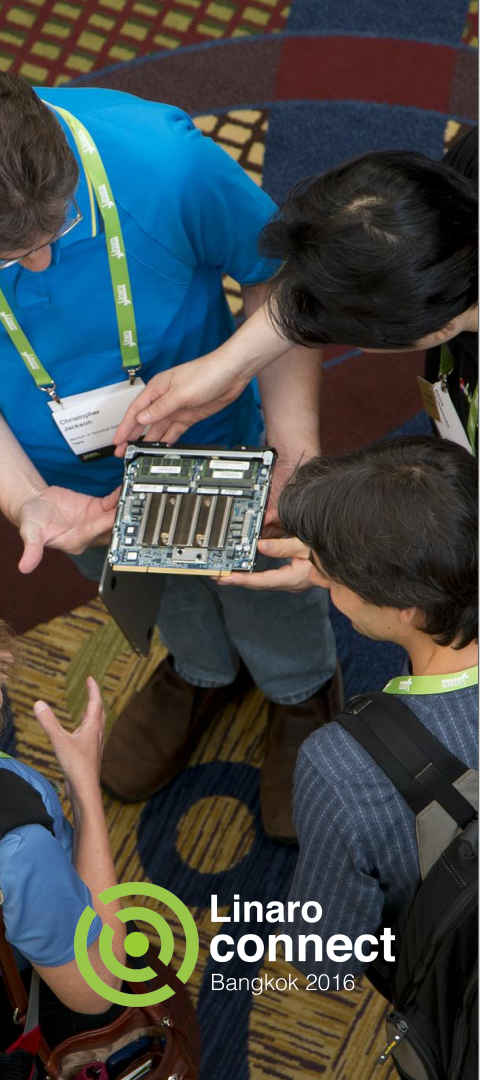
- default malloc
- scales well with many CPU cores



Replace the memory allocator

dlmalloc:

- much more efficient with low memory devices
- doesn’t scale well with many CPU cores
- likely to be removed for Android N

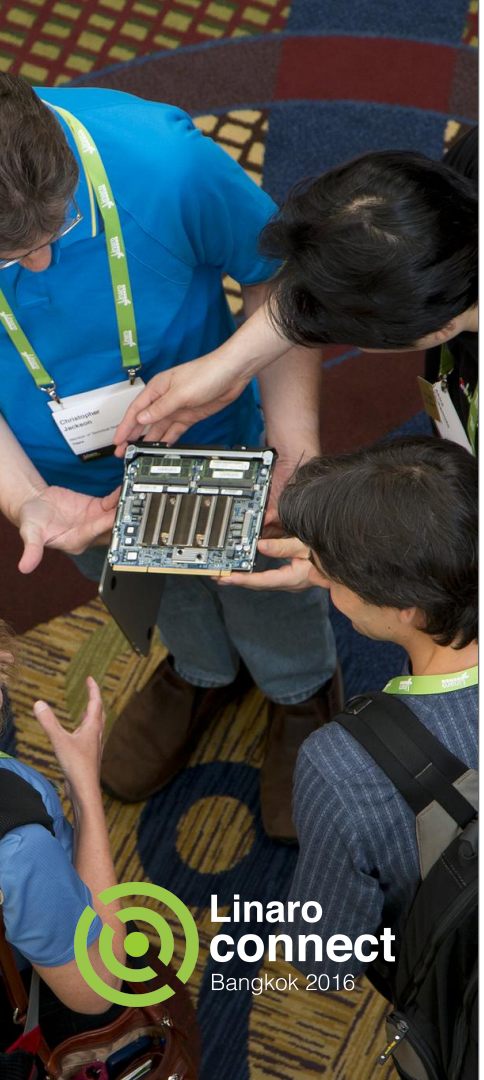


Replace the memory allocator

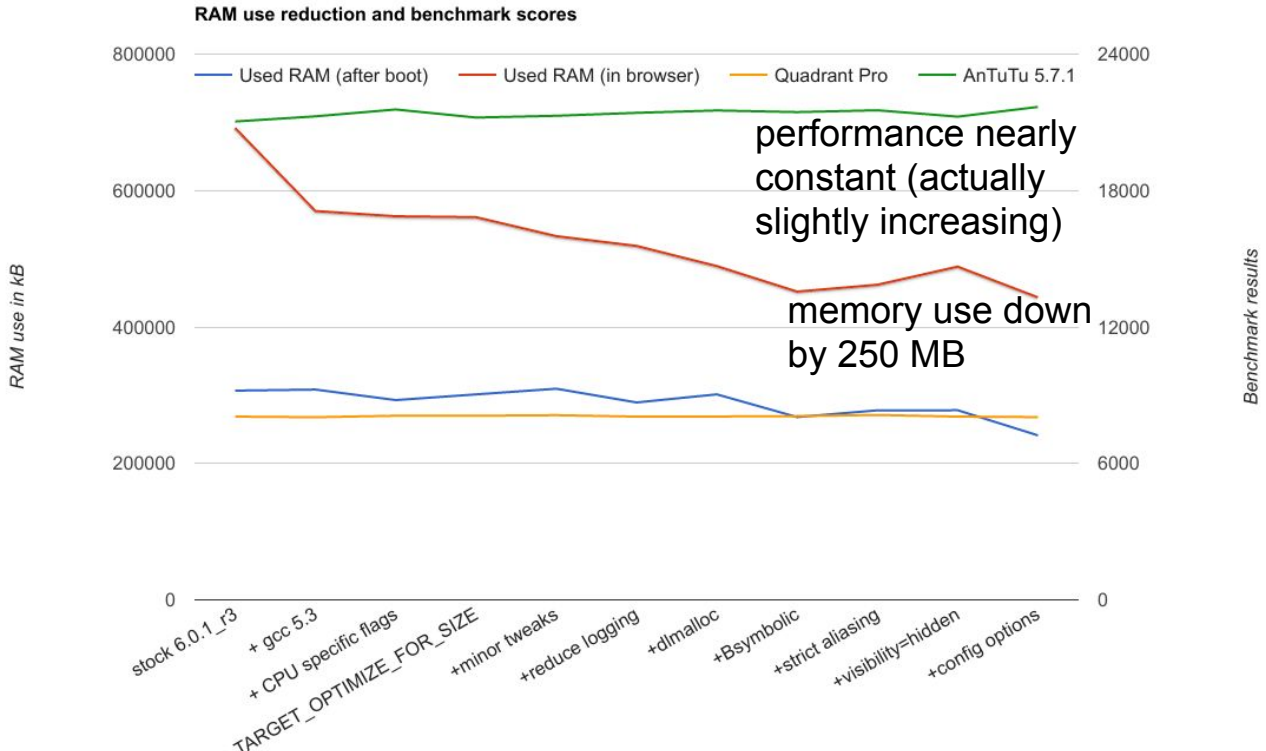
Is there a best of both worlds?

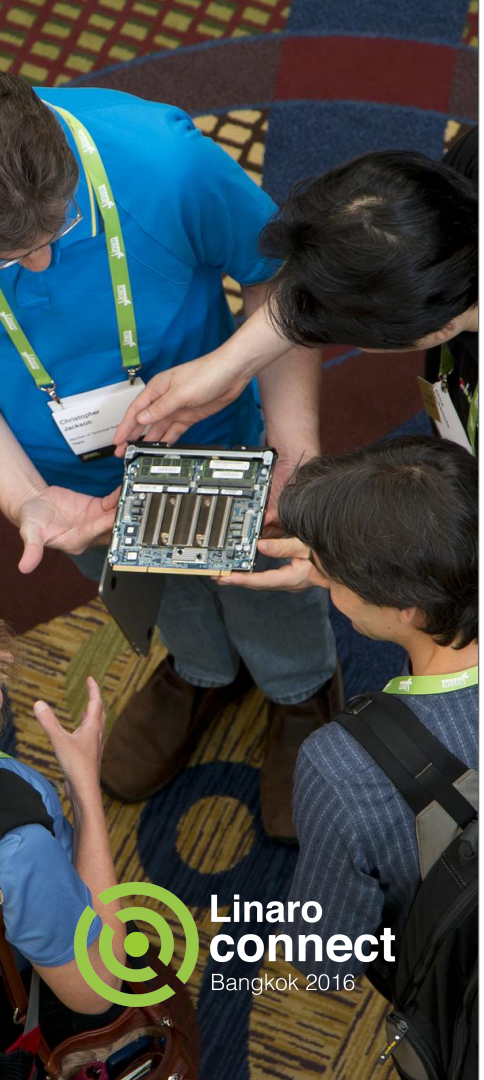
We've investigated jemalloc, dlmalloc, ptmalloc2, ptmalloc3.

Followup projects still in progress: TLSF malloc, tcmalloc, nedmalloc, musl's malloc (essentially dlmalloc algorithm with better locking)



Overall results

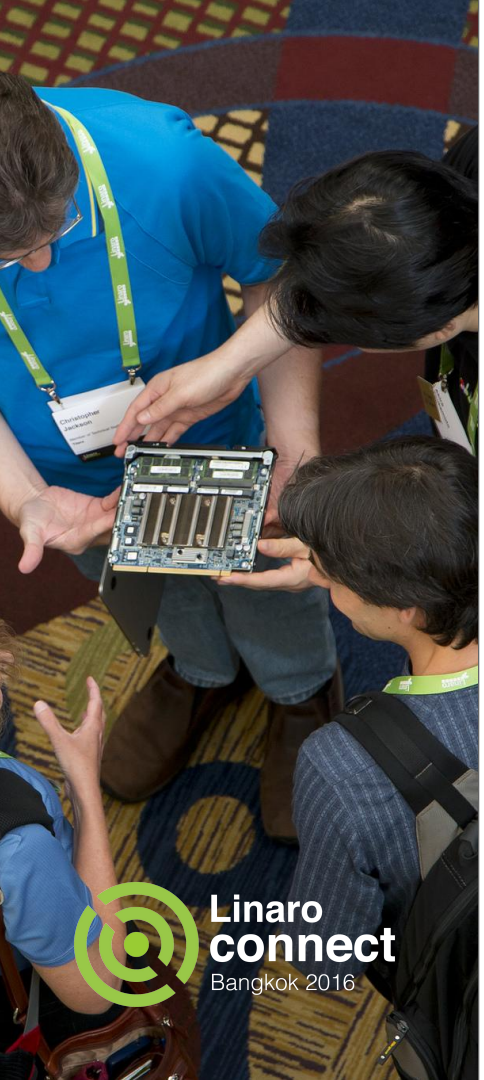




Results

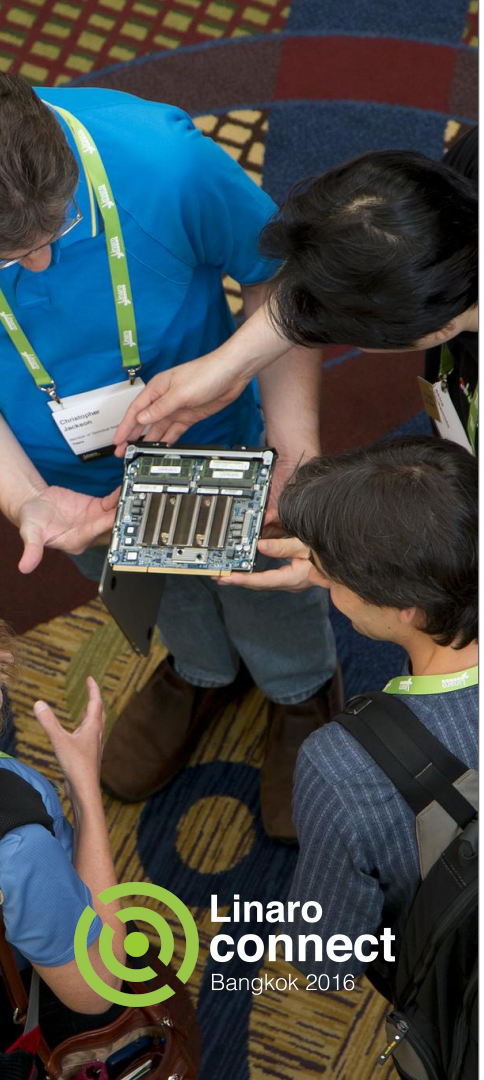
Toolchain features, malloc implementation, settings were all more effective than splitting libraries (which was most work).

Smaller changes can be more effective.



More room for optimizations

- Kernel features like zramfs and KSM (kernel samepage merging) have not been used in tests (postponed until new enough kernels can boot to UI on Nexus 7)
- SLOB allocator should probably be used in low-RAM kernels



More room for optimizations

- Investigate more malloc implementations
- Reimplement some Java based components in C/C++
- Check memory efficiency of various filesystems
- Repeat investigations in 64-bit world

Questions? Comments?

Talk to us now...

Or email:

bero@linaro.org

yongqin.liu@linaro.org