Compressed Swap for Embedded Linux

Alexander Belyakov, Intel Corp.

Outline.

1. Motivation
2. Underlying media types
3. Related works
4. MTD compression layer
   - driver place in kernel architecture
   - swap-in/out data flow
5. Performance expectations
6. User-mode performance
   - PCM
   - NAND
   - RAM
7. Conclusion

Motivation

- ➡ Memory requirements in embedded world constantly grow
- ➡ Most of embedded systems has both DRAM and flash memory

**Compress swapped pages
and store them to the flash.**

- ✚ More virtual memory available for applications
- ✚ More memory available for file-cache
- ✚ Power consumption is getting lower
- ✚ Entire design becomes cheaper
- ✚ Virtual memory subsystem becomes more healthy

- ✖ Performance degradation
- ✖ Erase cycles
- ✖ Wear out issues

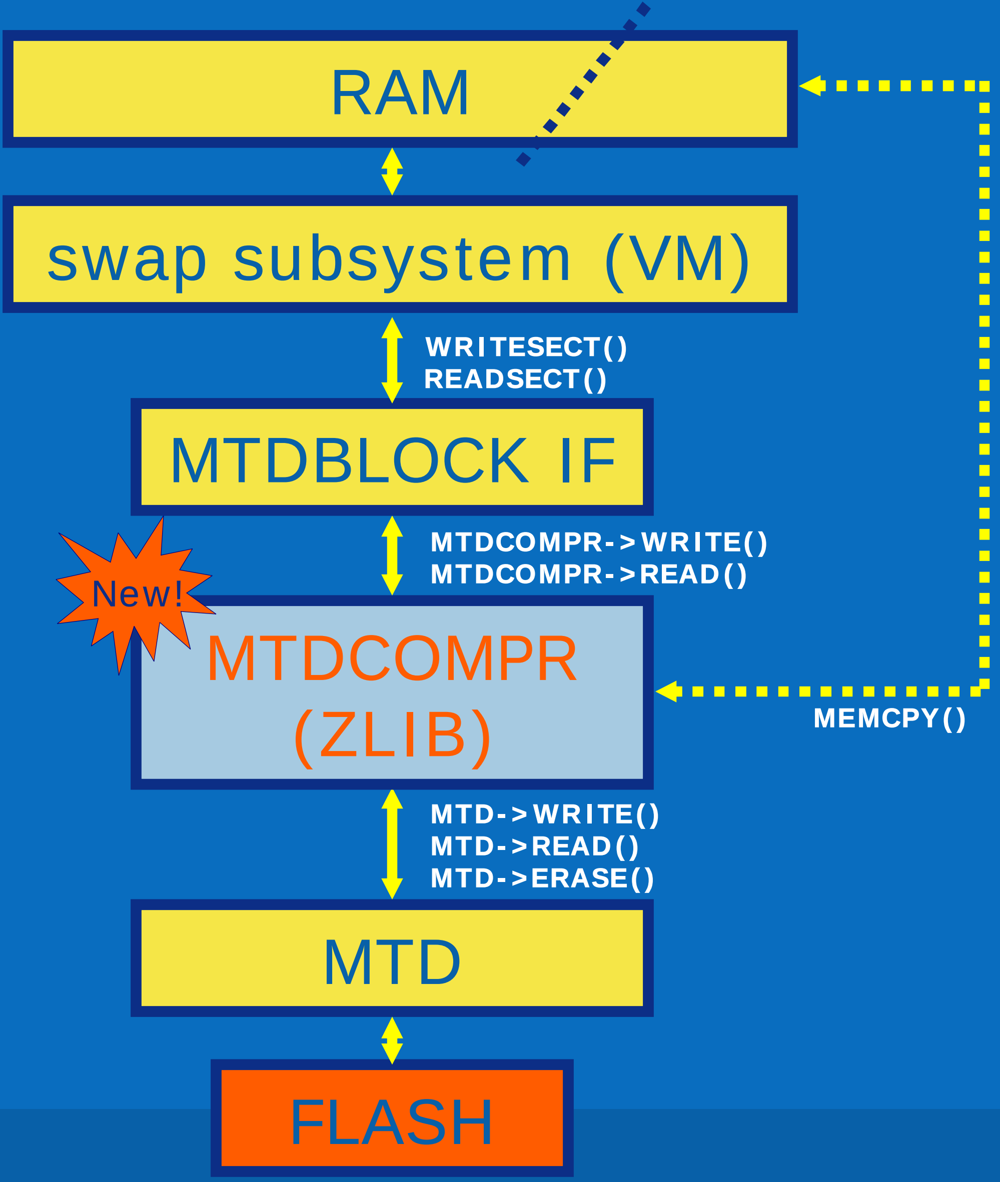
Underlying media types

1. NAND
   - ⊗ required explicit erase
   - ⊗ bad blocks
   - ⊗ wear out/leveling issues
   - ⊗ increased software complexity
2. PCM (phase change memory)
   - ✚ bit alterable writes
   - ✚ cycling endurance
   - ✚ performance
3. RAM
   - ✚ high performance
   - ✚ no hardware changes
   - ✚ extra virtual memory

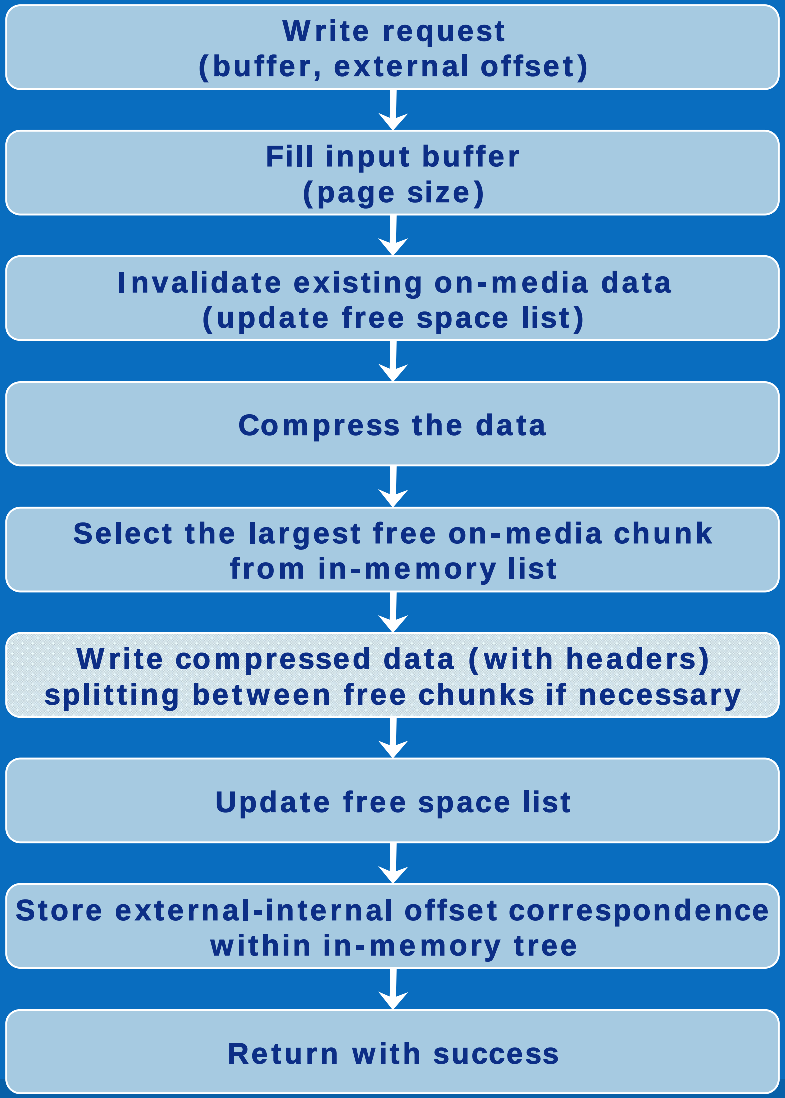
Related works

- hardware compressor/decompressor
  - *B. Tremaine, et al., “IBM memory expansion technology”*
- swap caching and compression
  - *T. Cortes, et al., “Improving Application Performance through Swap Compression”*
- compressed swap in RAM (embedded)
  - *Lei Yang, et al., “CRAMES: Compresses RAM for Embedded Systems”*
  - *Compressed Caching for Linux, http://code.google.com/p/compcache/*
- compressed swap on NAND (embedded)
  - *Sangduck Park, et al., “Compressed Swapping for NAND Flash Memory Based Embedded Systems”*

MTD Compression Layer (prototype)



MTDCOMPR swap-out data flow



Write request
(buffer, external offset)

Fill input buffer
(page size)

Invalidate existing on-media data
(update free space list)

Compress the data

Select the largest free on-media chunk
from in-memory list

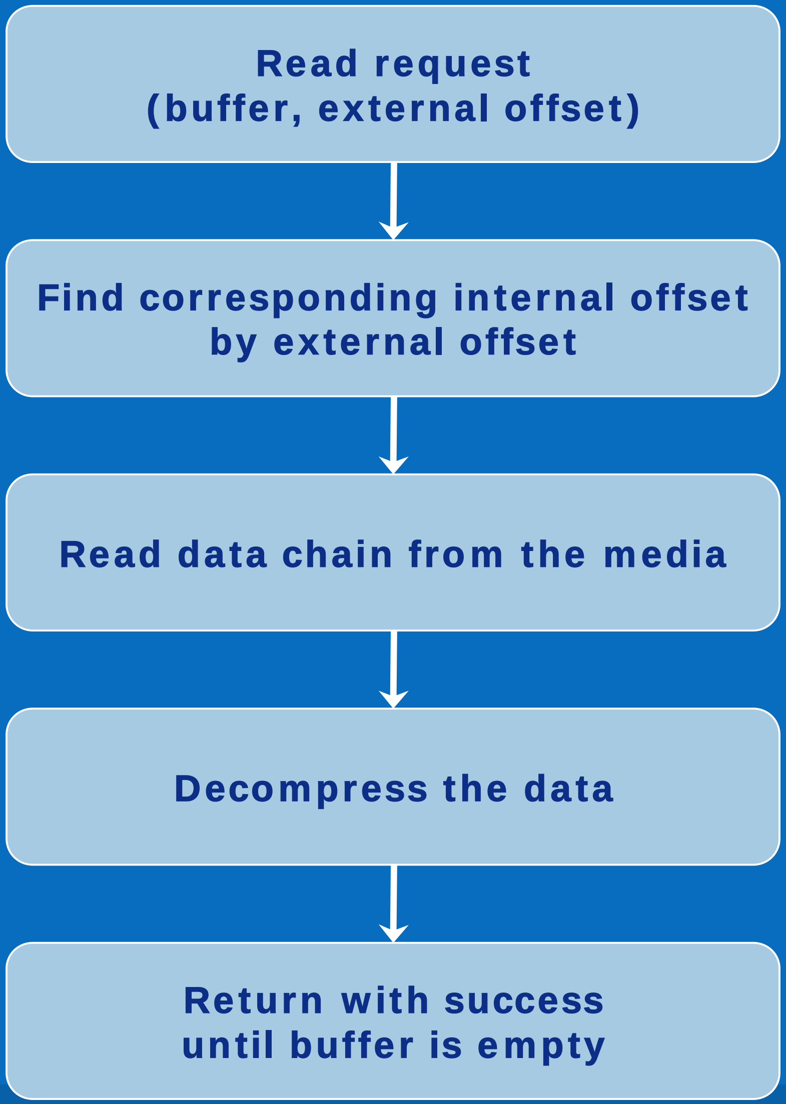
Write compressed data (with headers)
splitting between free chunks if necessary

Update free space list

Store external-internal offset correspondence
within in-memory tree

Return with success

MTDCOMPR swap-in data flow



Read request
(buffer, external offset)

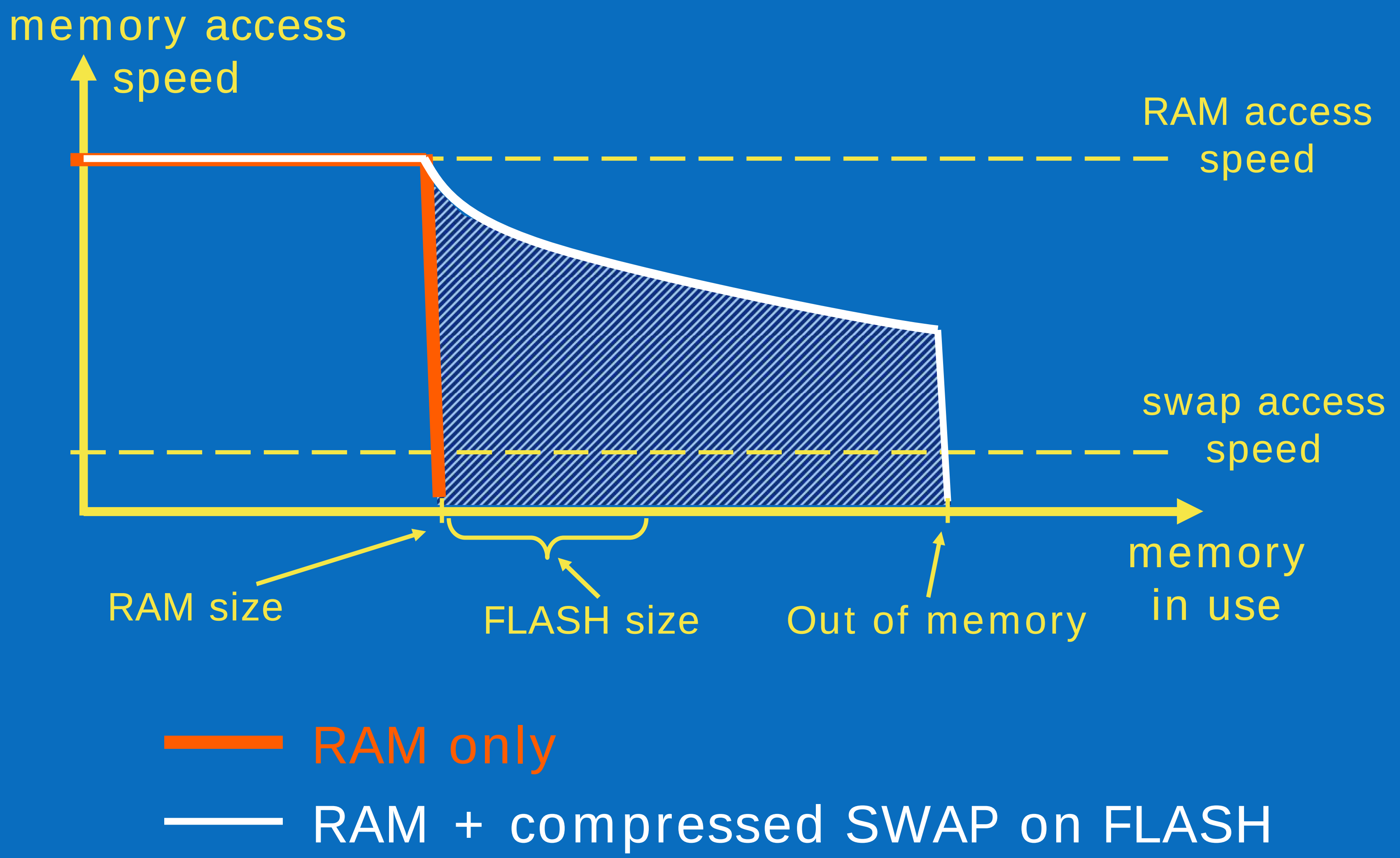
Find corresponding internal offset
by external offset

Read data chain from the media

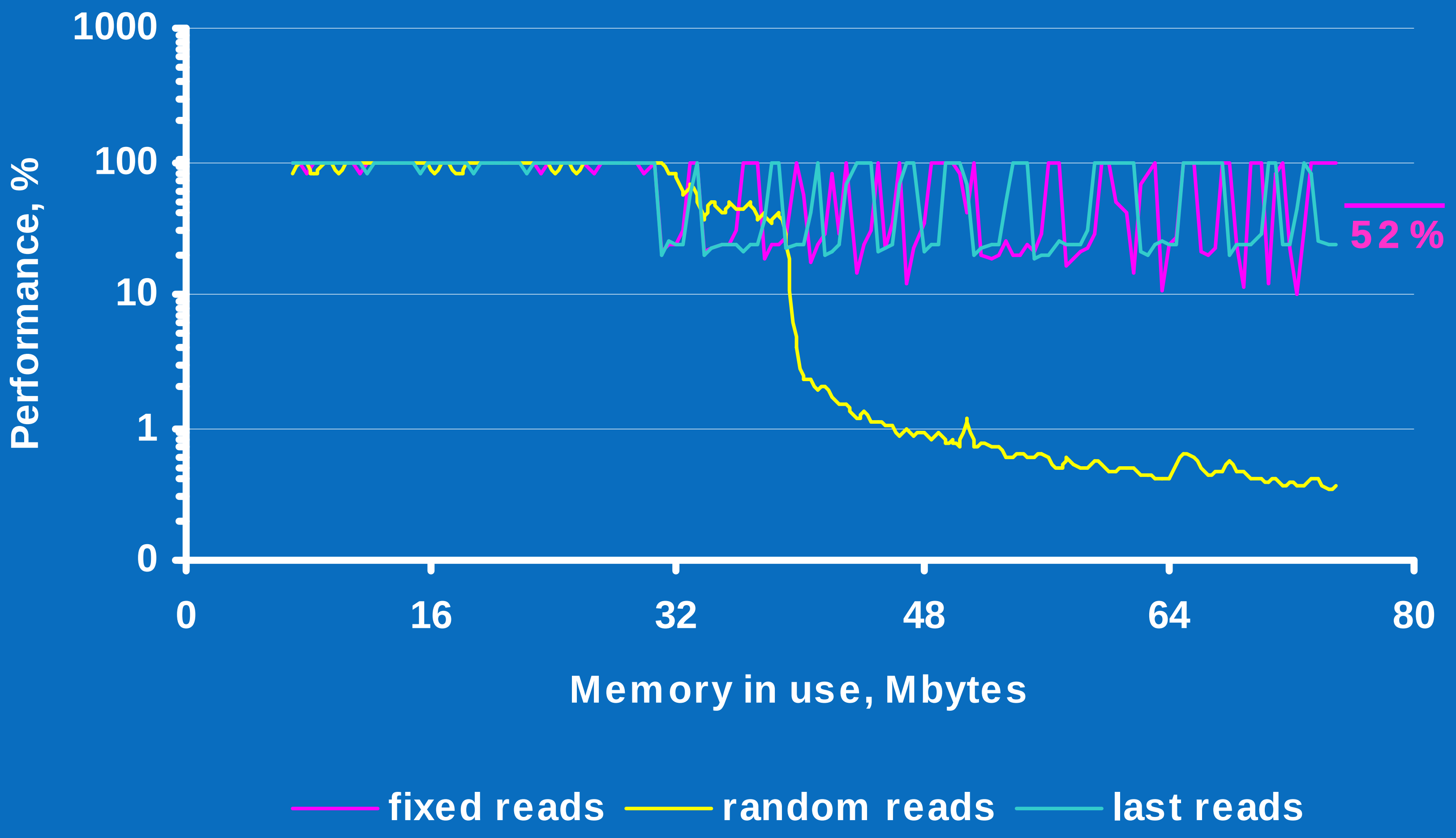
Decompress the data

Return with success
until buffer is empty

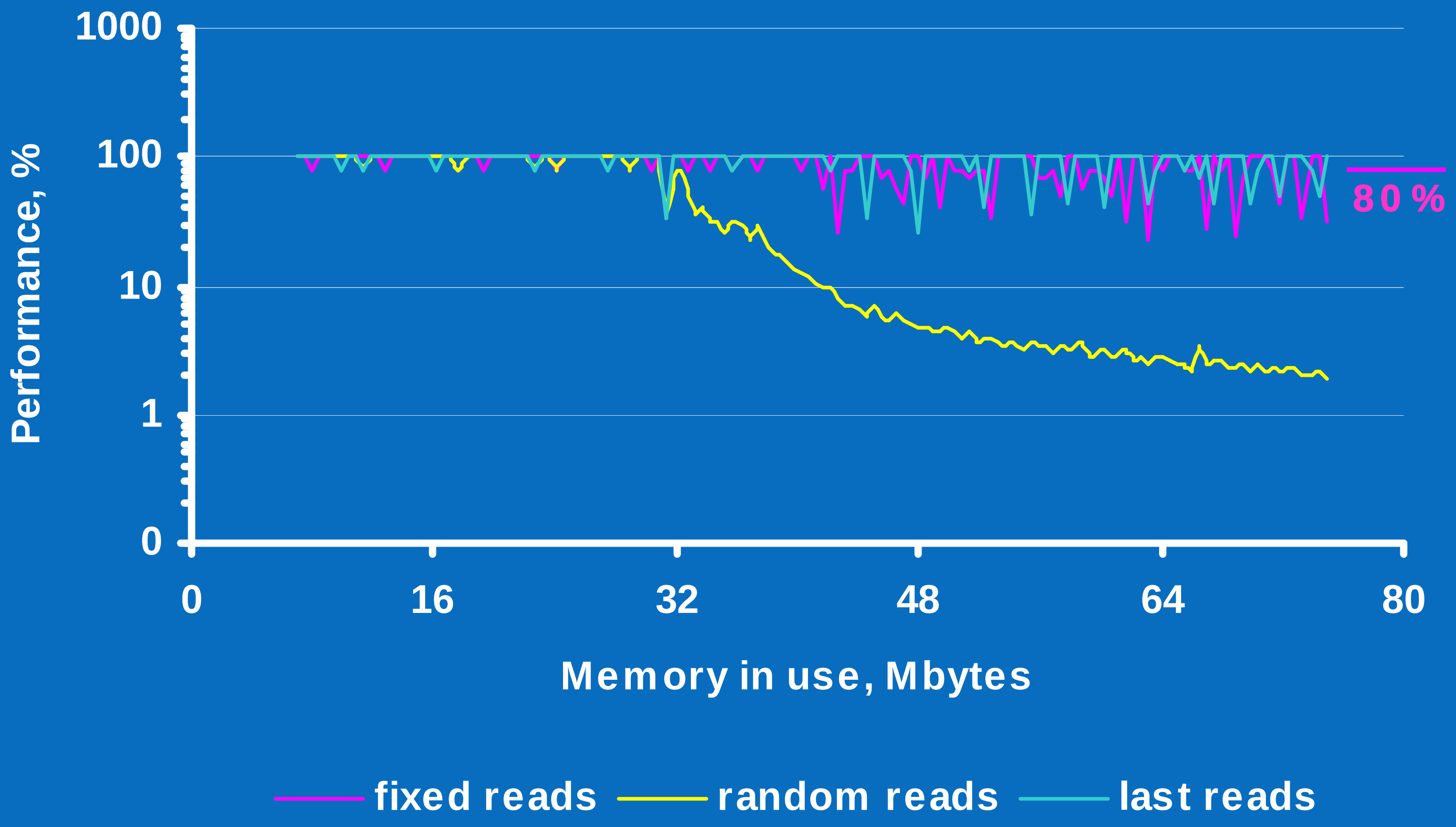
Performance expectations (flash)



User-mode performance (PXA271, 32MB RAM, 16MB NAND)

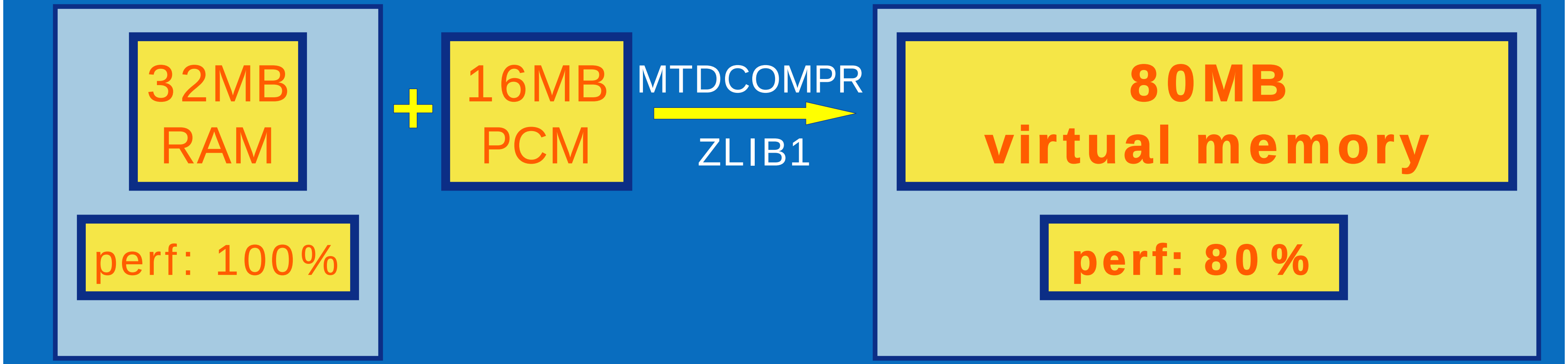


User-mode performance (PXA271, 32MB RAM, 16MB PCM)



Conclusion

- MTDCOMPR key features:
  - ✚ increase virtual memory
  - ✚ moderate performance impact
  - ✚ relatively simple
- Media types:
  - ✚ PCM – easy to use, hardware dependent
  - ✚ RAM – easy to use, no hardware dependence
  - ⊗ NAND – required complex solution, hardware dependent



32MB RAM

perf: 100%

+

16MB PCM

MTDCOMPR

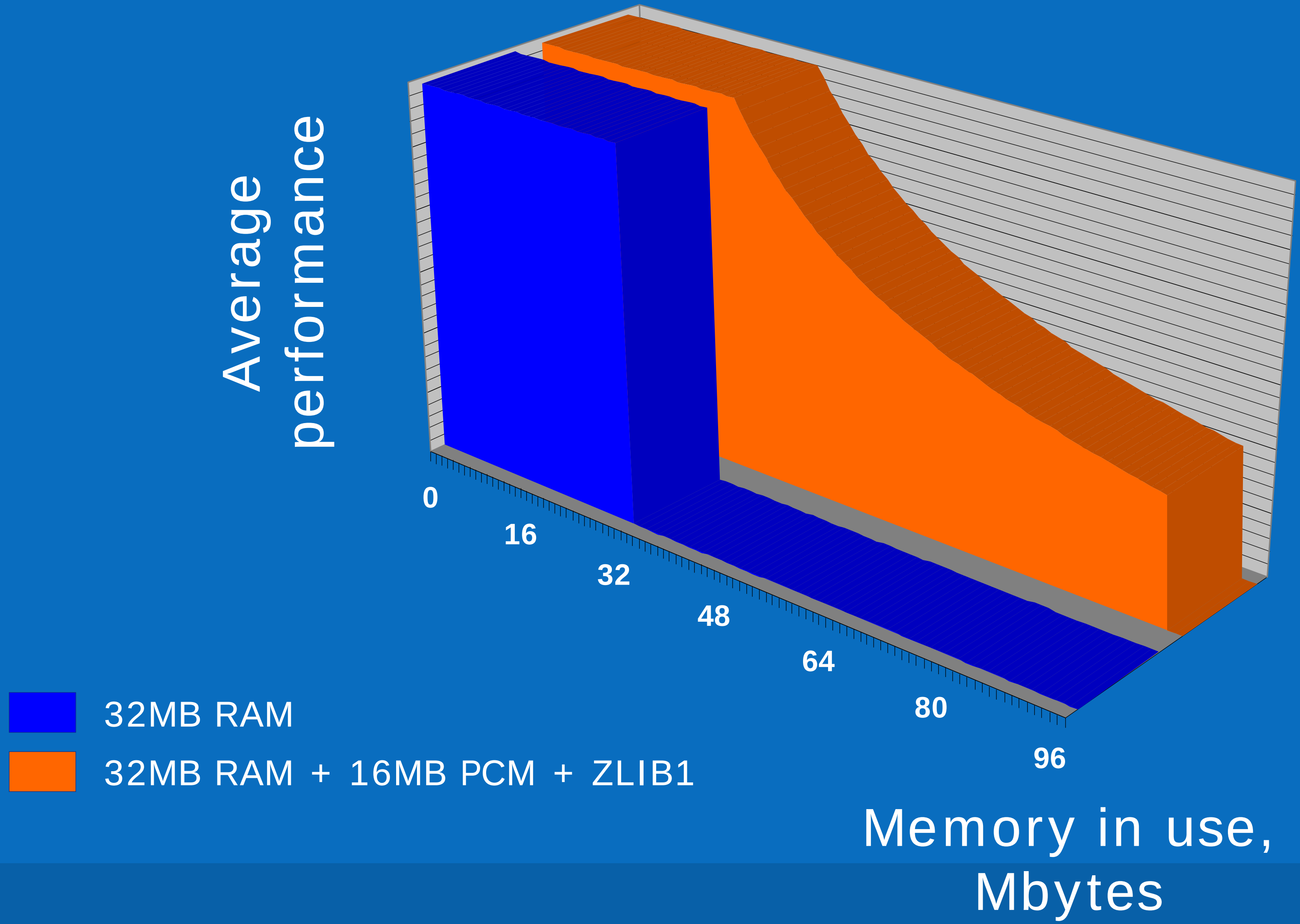
ZLIB1

80MB virtual memory

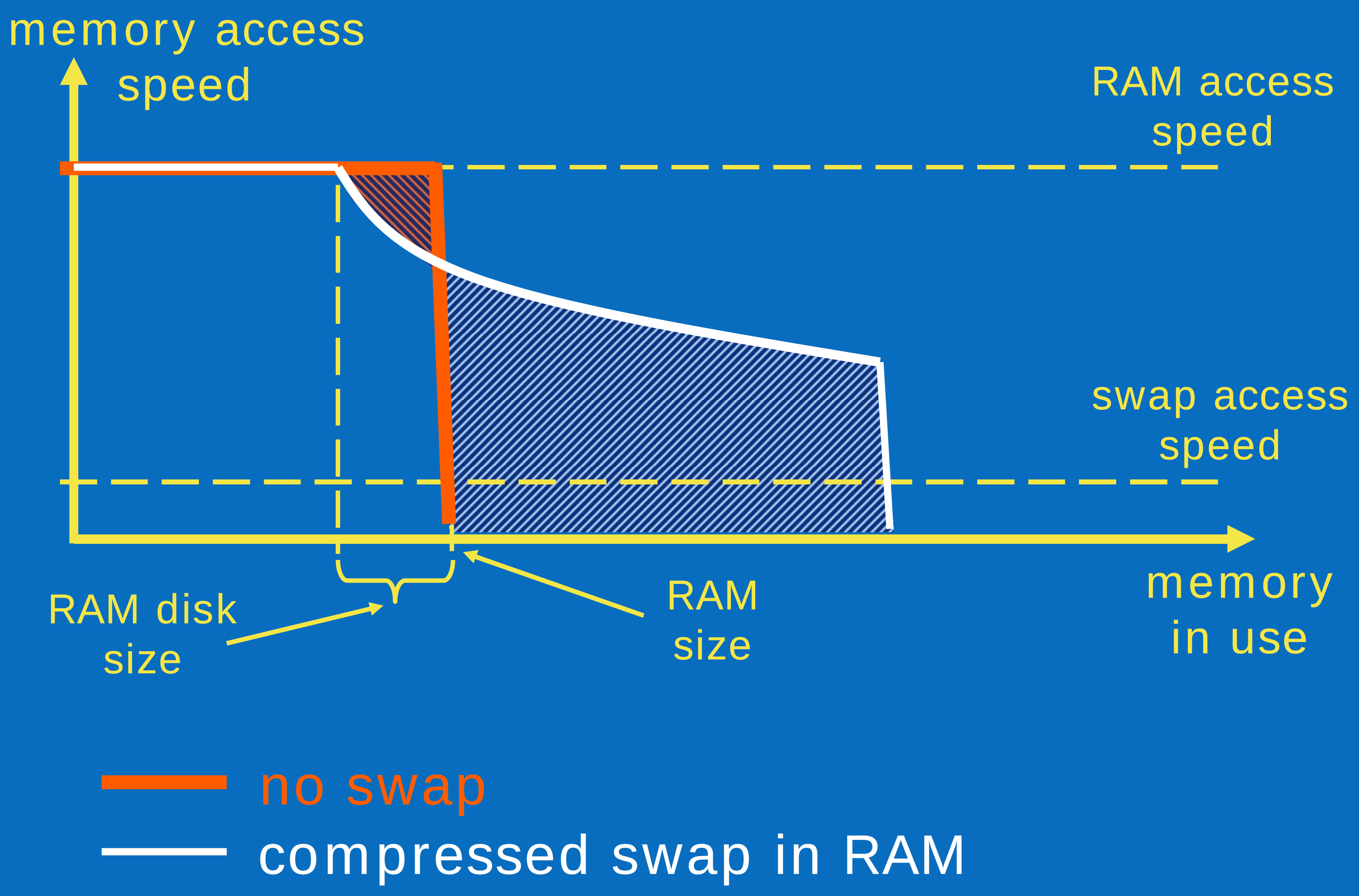
perf: 80%

Backup

Performance expectations (prototype based data)



Performance expectations (RAM)



Performance measurement setup

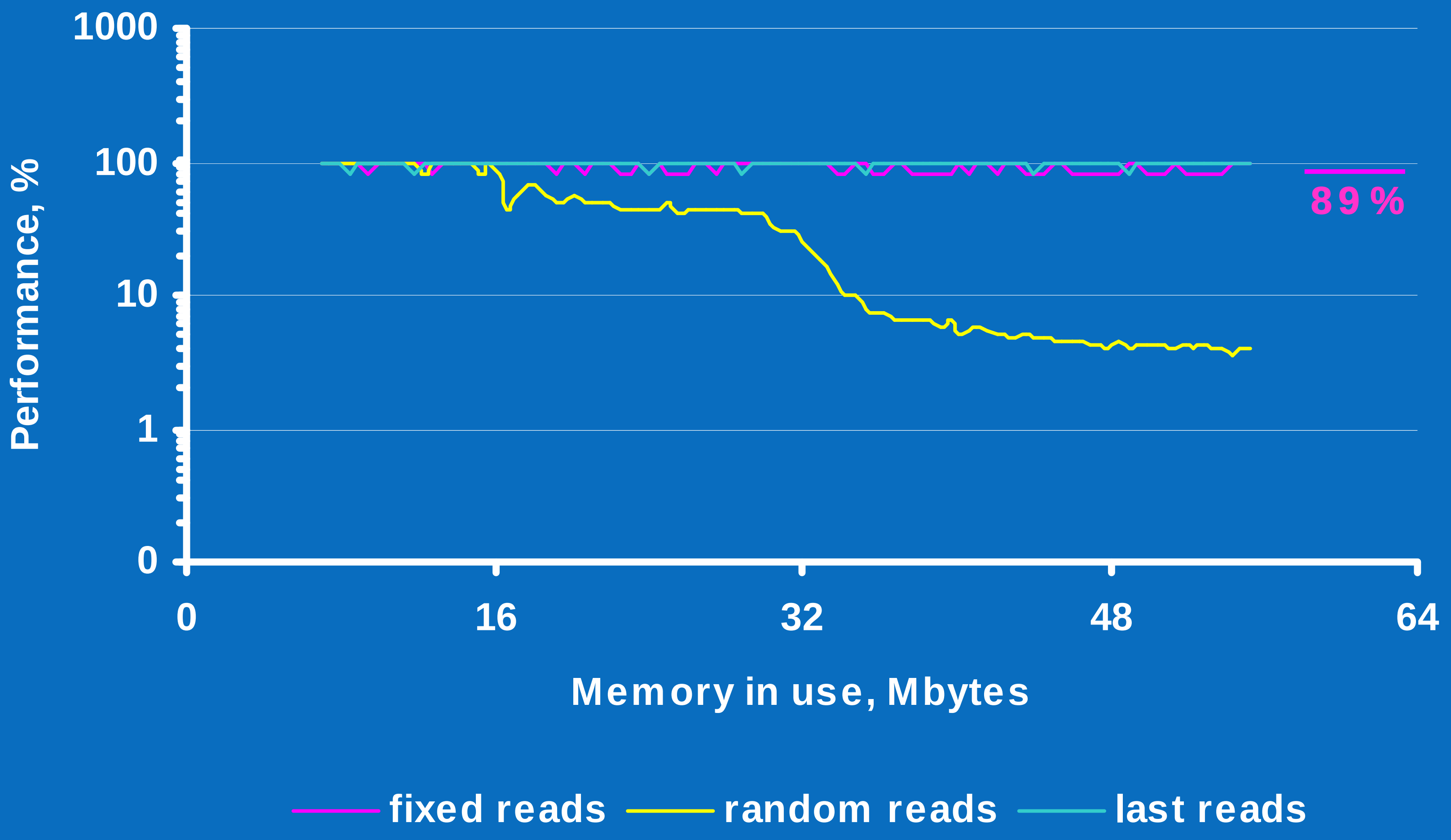
System:

- PXA271 (416 MHz) with 32 Mbytes of RAM (Mainstone II)
- Linux kernel 2.6.23.8
- rootfs on M18 (JFFS2)

Benchmarking application:

- Allocates memory chunk by chunk
- Fills it with data providing accurate compression ratio
- Accesses previously allocated chunks
  - random reads
  - fixed reads
  - last reads
- Measures performance depending on amount of allocated memory
- Deallocates memory

User-mode performance (PXA271, 32MB RAM)


Performance, %
Memory in use, Mbytes
fixed reads
random reads
last reads