OCTO$^+$: Optimized Checkpointing of B$^+$ Trees for Non-Volatile Main Memory Wear-Leveling

Christian Hakert, Roland Kühn, Kuan-Hsun Chen, Jens Teubner, Jian-Jia Chen

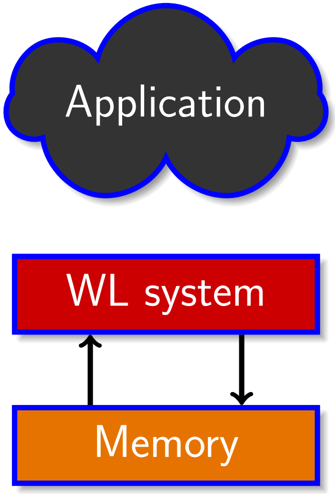
Department of Computer Science, Chair 12 / Chair 6
TU Dortmund University, Germany

August, 19 2021

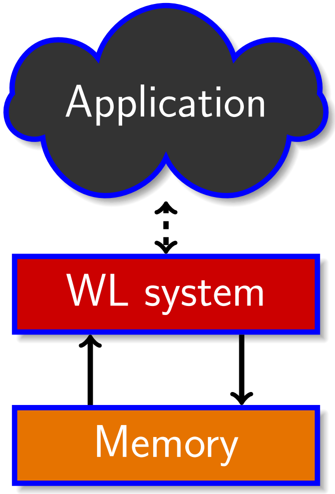
The 10th IEEE Non-Volatile Memory Systems and Applications Symposium August 18-20, 2021, Virtual Conference

This research is supported in parts by the German Research Foundation (DFG) as part of OneMemory and SFB 876, subprojects A1 and A2

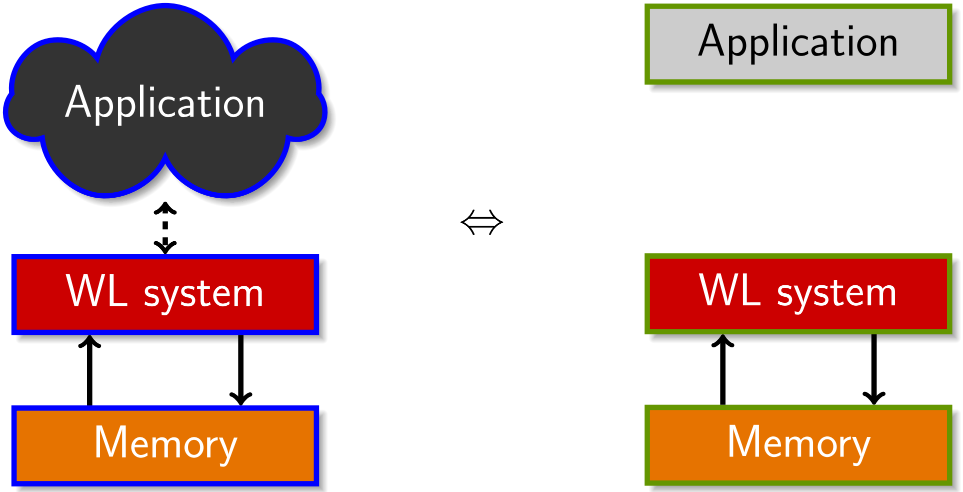
Wear-Leveling and Applications



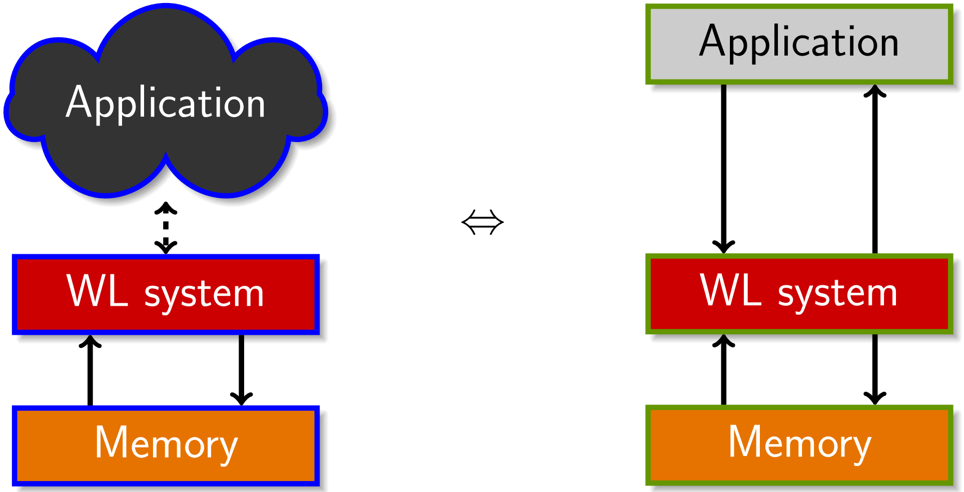
Wear-Leveling and Applications



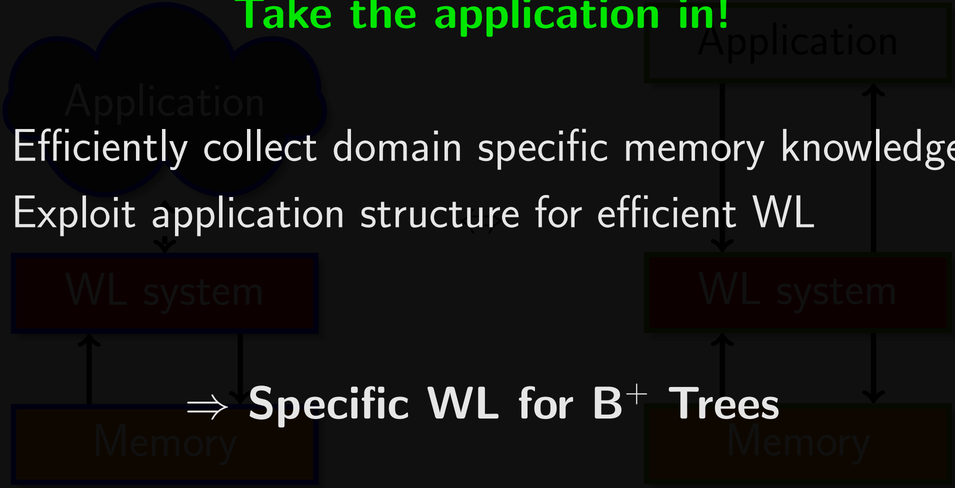
Wear-Leveling and Applications



Wear-Leveling and Applications



Wear-Leveling and Applications



Take the application in!

- Efficiently collect domain specific memory knowledge
- Exploit application structure for efficient WL

$\Rightarrow$ **Specific WL for B$^+$ Trees**

Outline

System Setup

Iterative write scheme:

- Popular for PCM
- Sense cell before update
- $\Rightarrow$ no write on unchanged value

$$\boxed{00000000} \rightsquigarrow \boxed{11111111}$$

vs.

$$\boxed{00101100} \rightsquigarrow \boxed{00111100}$$
$$\rightsquigarrow \boxed{00111110} \rightsquigarrow \boxed{00111111}$$

Expensive data migration!

System Setup

Iterative write scheme:

- Popular for PCM
- Sense cell before update
- $\Rightarrow$ no write on unchanged value

$$\boxed{00000000} \rightsquigarrow \boxed{11111111}$$

vs.

$$\boxed{00101100} \rightsquigarrow \boxed{00111100}$$
$$\rightsquigarrow \boxed{00111110} \rightsquigarrow \boxed{00111111}$$

Expensive data migration!

Checkpointing:

- Keep data in VM
- Regularly copy *checkpoint* to NVM
  - Utilize non-volatiliy + low DRAM latency

System Setup

Iterative write scheme:

- Popular for PCM
- Sense cell before update
- $\Rightarrow$ no write on unchanged value

$$\boxed{00000000} \rightsquigarrow \boxed{11111111}$$

vs.

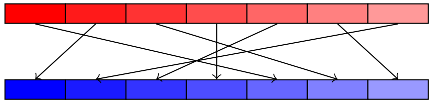
$$\boxed{00101100} \rightsquigarrow \boxed{00111100} \rightsquigarrow \boxed{00111110} \rightsquigarrow \boxed{00111111}$$

Expensive data migration!

Checkpointing:

- Keep data in VM
- Regularly copy *checkpoint* to NVM
  - Utilize non-volatiliy + low DRAM latency

Mapping between VM memory and NVM checkpoint:



System Setup

Iterative write scheme:

- Popular for PCM
- Sense cell before update
- $\Rightarrow$ no write on unchanged value

$$\boxed{00000000} \rightsquigarrow \boxed{11111111}$$

vs.

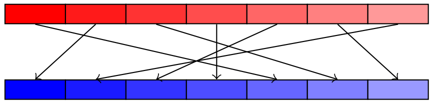
$$\boxed{00101100} \rightsquigarrow \boxed{00111100}$$
$$\rightsquigarrow \boxed{00111110} \rightsquigarrow \boxed{00111111}$$

Expensive data migration!

Checkpointing:

- Keep data in VM
- Regularly copy *checkpoint* to NVM
  - Utilize non-volatiliy + low DRAM latency

Mapping between VM memory and NVM checkpoint:



Easy to modify between checkpoints!

System Setup

Iterative write scheme:

- Popular for PCM
- Sense cell before update
- $\Rightarrow$ no write on unchanged value

$\boxed{00000000} \rightsquigarrow \boxed{11111111}$

vs.

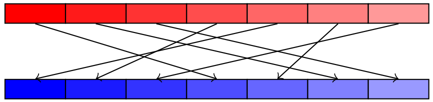
$\boxed{00101100} \rightsquigarrow \boxed{00111100}$
$\rightsquigarrow \boxed{00111110} \rightsquigarrow \boxed{00111111}$

Expensive data migration!

Checkpointing:

- Keep data in VM
- Regularly copy *checkpoint* to NVM
  - Utilize non-volatiliy + low DRAM latency

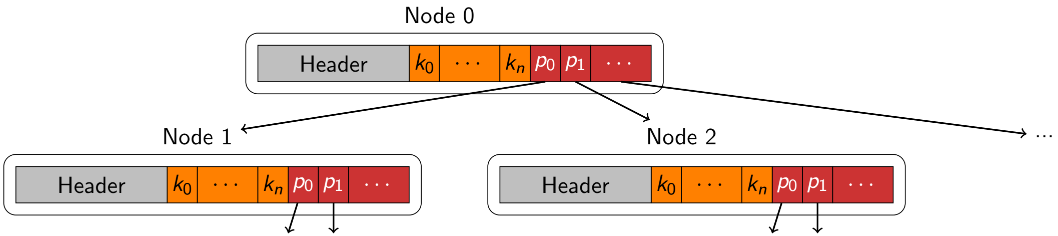
Mapping between VM memory and NVM checkpoint:



Easy to modify between checkpoints!

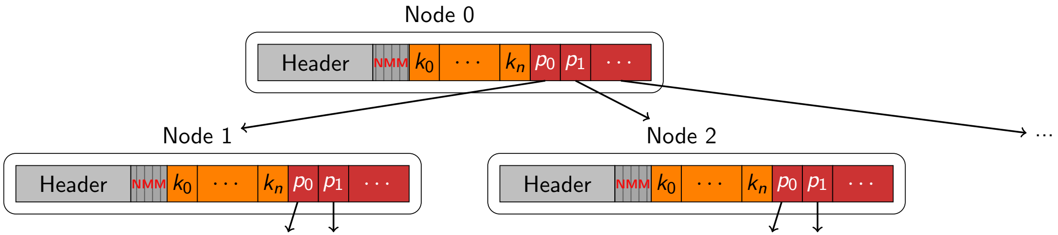
B^+ Trees / Write Information Collection

- Sorted split values and child pointers
- B^+ Tree nodes consume one NVM block while checkpointing



B^+ Trees / Write Information Collection

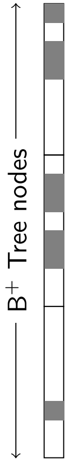
- Sorted split values and child pointers
- B^+ Tree nodes consume one NVM block while checkpointing



- Tree modification knows changed keys / pointers
- Maintain a **Node Modification Mask (NMM)**
  - Bitmask indicating modified parts of nodes
  - Reset at checkpoints

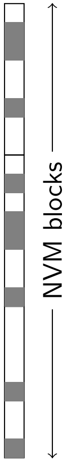
OCTO$^+$ Wear-Leveling

Abstract logical (application) memory and physical memory:



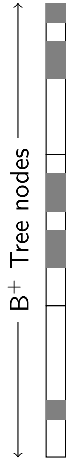
Keep track of:

- **B$^+$ Node Modification (NMM)**
  - Short term usage of logic memory
- **NVM Block Age**
  - Long term utilization of physical memory
  - Accumulation of mapped B$^+$ Tree NMMs



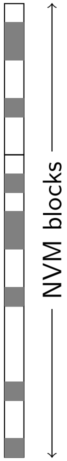
OCTO$^+$ Wear-Leveling

Abstract logical (application) memory and physical memory:



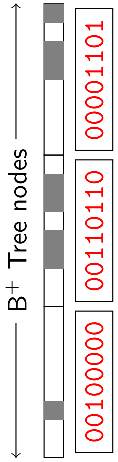
Keep track of:

- **B$^+$ Node Modification (NMM)**
  - Short term usage of logic memory
- **NVM Block Age**
  - Long term utilization of physical memory
  - Accumulation of mapped B$^+$ Tree NMMs
- Encode both as bitmasks
  - 1 if region was modified since the last checkpoint
  - 1 if accumulated count of region is higher than the average within the block (+threshold)



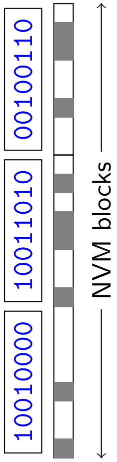
OCTO$^+$ Wear-Leveling

Abstract logical (application) memory and physical memory:



Keep track of:

- **B^+ Node Modification (NMM)**
  - Short term usage of logic memory
- **NVM Block Age**
  - Long term utilization of physical memory
  - Accumulation of mapped B^+ Tree NMMs
- Encode both as bitmasks
  - 1 if region was modified since the last checkpoint
  - 1 if accumulated count of region is higher than the average within the block (+threshold)



OCTO$^+$ Wear-Leveling (cont.)

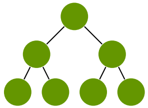
Intra-Block WL:

1. Start from the current checkpoint mapping, build bitmaps
2. Release all NVM blocks, which are not 00000000 $\rightarrow$ results in n free NVM blocks and n unmapped B$^+$ tree nodes
3. Re-Shuffle the mapping:

OCTO$^+$ Wear-Leveling (cont.)

Intra-Block WL:

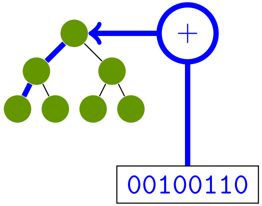
1. Start from the current checkpoint mapping, build bitmaps
2. Release all NVM blocks, which are not 00000000 $\rightarrow$ results in n free NVM blocks and n unmapped B$^+$ tree nodes
3. Re-Shuffle the mapping:



OCTO$^+$ Wear-Leveling (cont.)

Intra-Block WL:

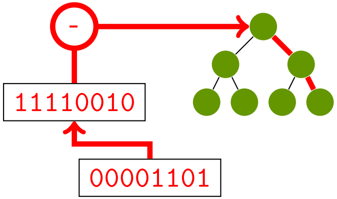
1. Start from the current checkpoint mapping, build bitmaps
2. Release all NVM blocks, which are not 00000000 $\rightarrow$ results in n free NVM blocks and n unmapped B$^+$ tree nodes
3. Re-Shuffle the mapping:



OCTO$^+$ Wear-Leveling (cont.)

Intra-Block WL:

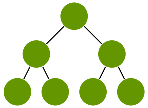
1. Start from the current checkpoint mapping, build bitmaps
2. Release all NVM blocks, which are not 00000000 $\rightarrow$ results in n free NVM blocks and n unmapped B$^+$ tree nodes
3. Re-Shuffle the mapping:



OCTO$^+$ Wear-Leveling (cont.)

Intra-Block WL:

1. Start from the current checkpoint mapping, build bitmaps
2. Release all NVM blocks, which are not 00000000 $\rightarrow$ results in n free NVM blocks and n unmapped B$^+$ tree nodes
3. Re-Shuffle the mapping:



Inter-Block WL:

1. Track youngest and oldest NVM block (min / max amount of writes to subblocks)
2. If both have uneven intra block WL, exchange logic mapping
3. Youngest / Oldest block are excluded from subsequent checkpoint remapping

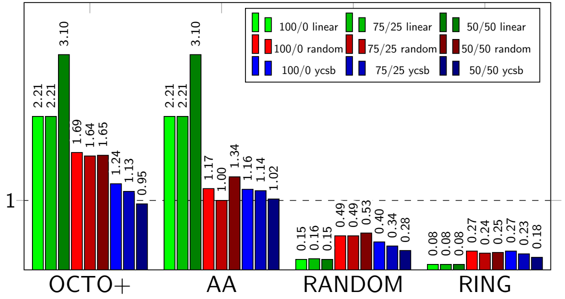
Evaluation

- Execute full implementation in full system simulation
- 3 B$^+$ Tree benchmarks (+3 insert/update ratios)

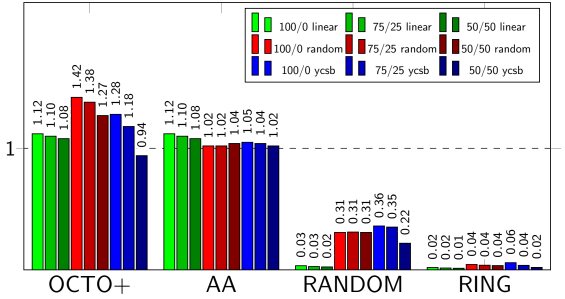
Evaluation

- Execute full implementation in full system simulation
- 3 B^+ Tree benchmarks (+3 insert/update ratios)

Lifetime Improvement (small - 20k Ops.)



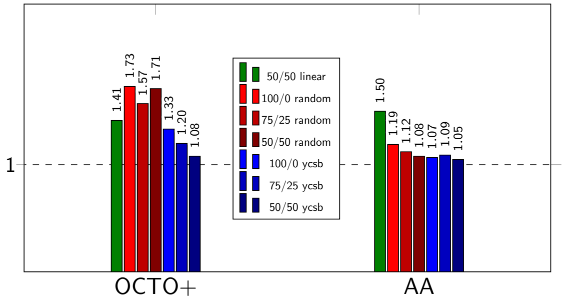
Wear-Leveling Potential (small - 20k Ops.)



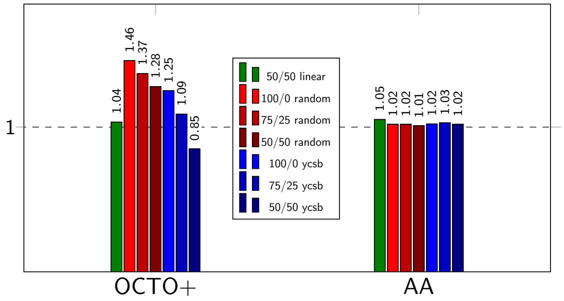
Evaluation

- Execute full implementation in full system simulation
- 3 B^+ Tree benchmarks (+3 insert/update ratios)

Lifetime Improvement (big - 50k Ops.)



Wear-Leveling Potential (big - 50k Ops.)



Takeaways

- Applications rarely explicitly taken into wear-leveling

Takeaways

- Applications rarely explicitly taken into wear-leveling
- Extend the application itself to track aging information

Takeaways

- Applications rarely explicitly taken into wear-leveling
- Extend the application itself to track aging information
- Hook into checkpointing for low overhead wear-leveling

Takeaways

- Applications rarely explicitly taken into wear-leveling
- Extend the application itself to track aging information
- Hook into checkpointing for low overhead wear-leveling

$\Rightarrow$ Careful wear-leveling for iterative write schemes required

$\Rightarrow$ Improved memory lifetime and aided wear-leveling beyond application scope

Takeaways

- Applications rarely explicitly taken into wear-leveling
- Extend the application itself to track aging information
- Hook into checkpointing for low overhead wear-leveling

$\Rightarrow$ Careful wear-leveling for iterative write schemes required

$\Rightarrow$ Improved memory lifetime and aided wear-leveling beyond application scope

Thank You!

Questions? $\Rightarrow$ `christian.hakert@tu-dortmund.de` / `roland.kuehn@cs.tu-dortmund.de`