# Towards Virtual Machine Image Management for Persistent Memory

MSST 2019


**Jiachen Zhang**, Lixiao Cui, Peng Li, Xiaoguang Liu, Gang Wang
Nankai-Baidu Joint Lab, Nankai University

Parallel and Distributed Software Technology Lab | Nankai - Baidu Joint Laboratory

# Agenda

- Background & Motivation
- Design & Optimization
- Performance Evaluation

# Agenda

- Background & Motivation
- Design & Optimization
- Performance Evaluation

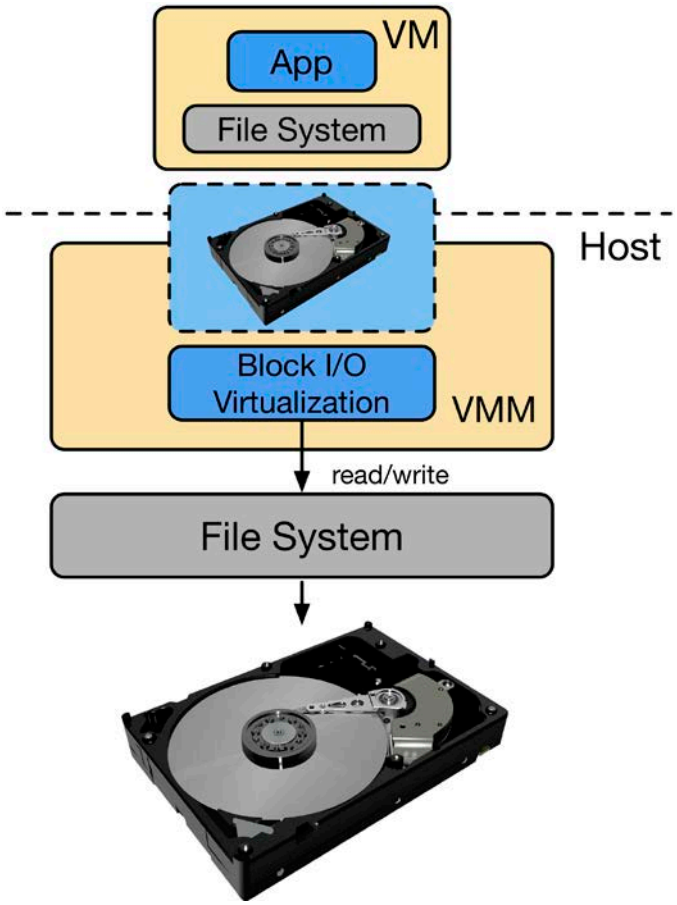
## What is Persistent Memory (PM)?

- DIMM form device based Non-Volatile Memory (NVM) technologies.
- Also known as Storage Class Memory (SCM).
- Compared with DRAM :
  - Higher capacity
  - Non-volatile data storage
- Compared with external block storage:
  - Byte-addressable
  - Ultra-low latency ( <1 us )

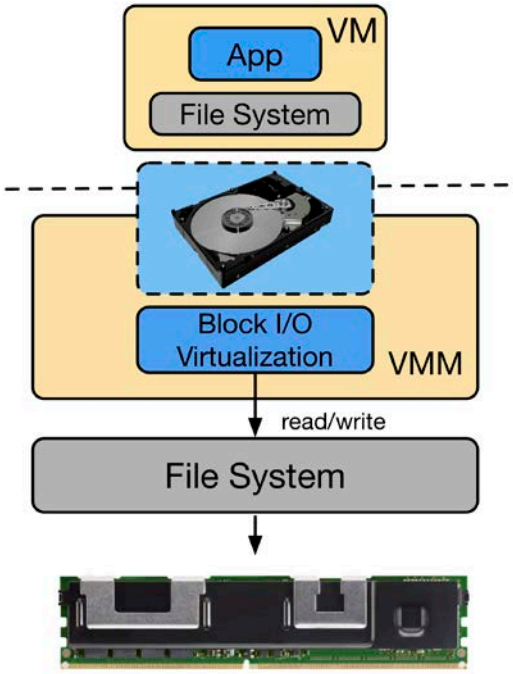
Intel’s DIMM form persistent memory

## Block Storage Virtualization

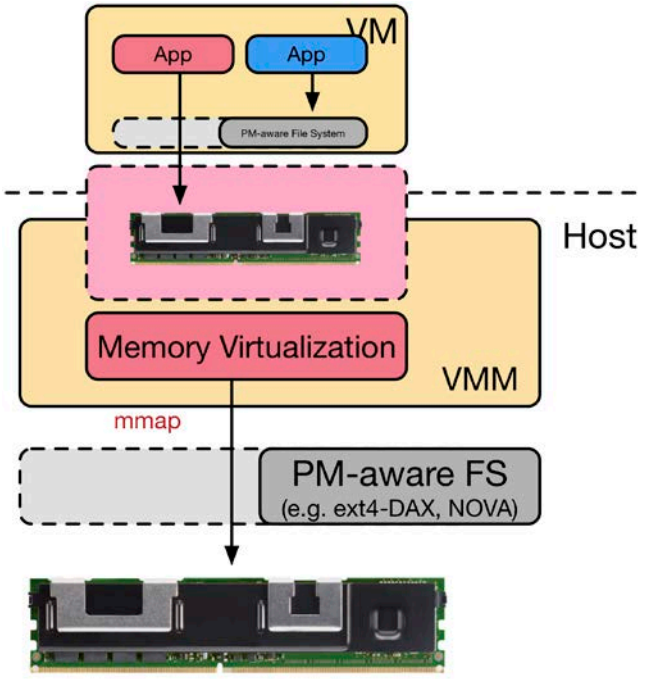
- Virtual Machine Monitor (VMM) emulate a virtual disk inside the virtual machine.
- Virtual disk is backed by an image file created on the host file system.
- Virtual disk emulation and image file management are handled by VMM's block I/O virtualization mechanism.



## PM Device Virtualization



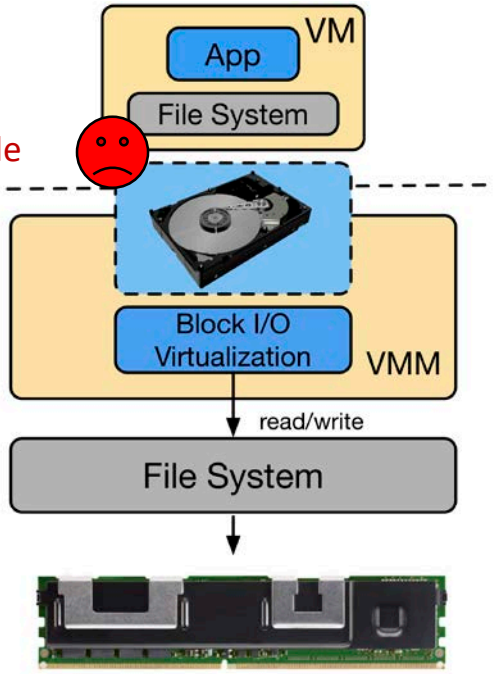

I/O virtualization




Memory virtualization

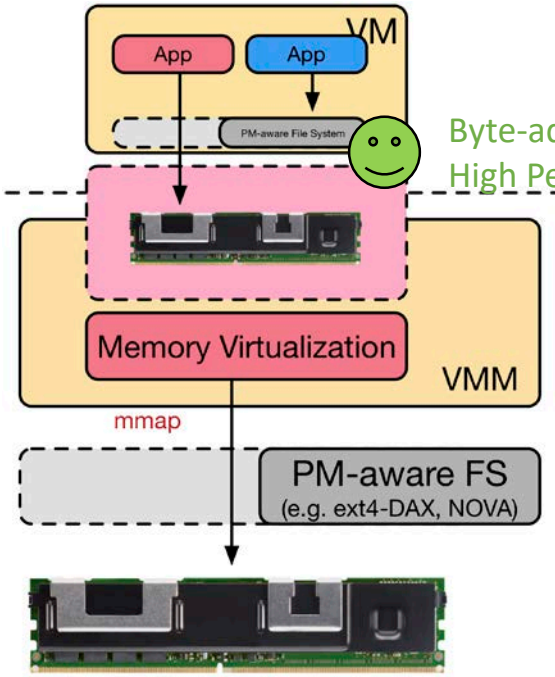
## PM Device Virtualization

Not byte-addressable
512 B granularity




I/O virtualization

Byte-addressable &
High Performance




Memory virtualization

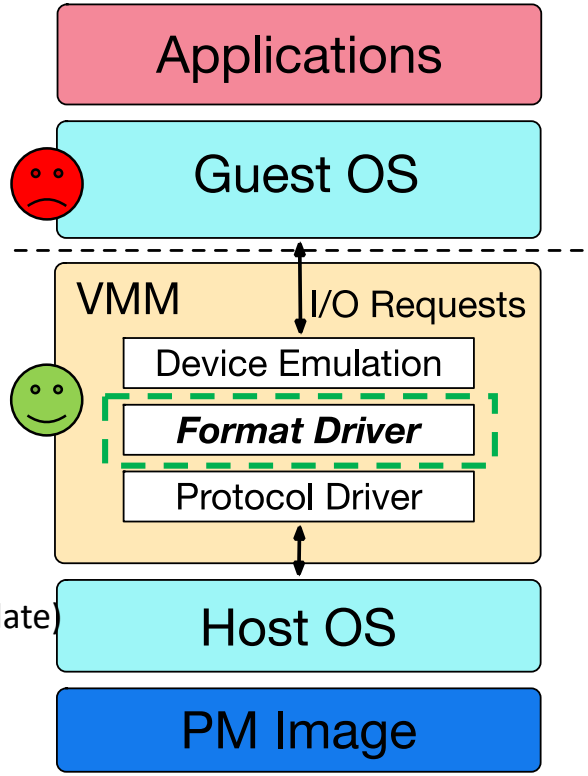
Which one should we choose?

## Data Access Path of the Two Mechanisms

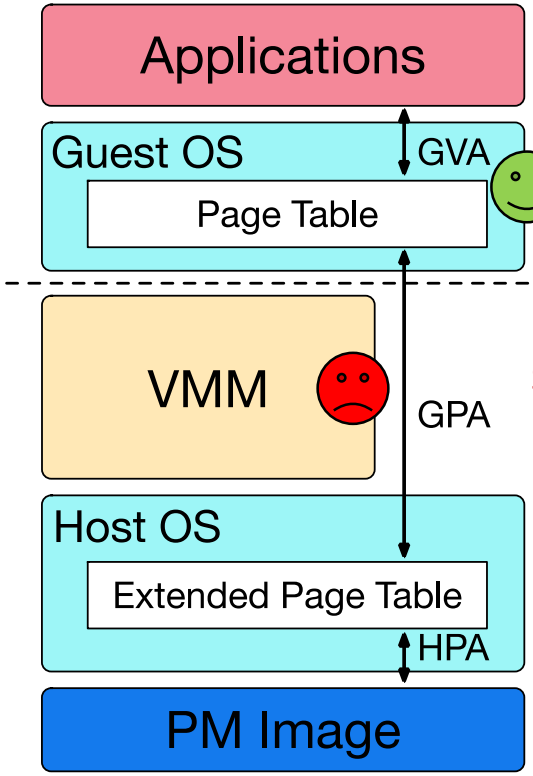
Not byte-addressable

Storage virtualization features:
(e.g. qcow2)

- thin-provision
- snapshot
- Base image (template)




QEMU's Block IO Virtualization



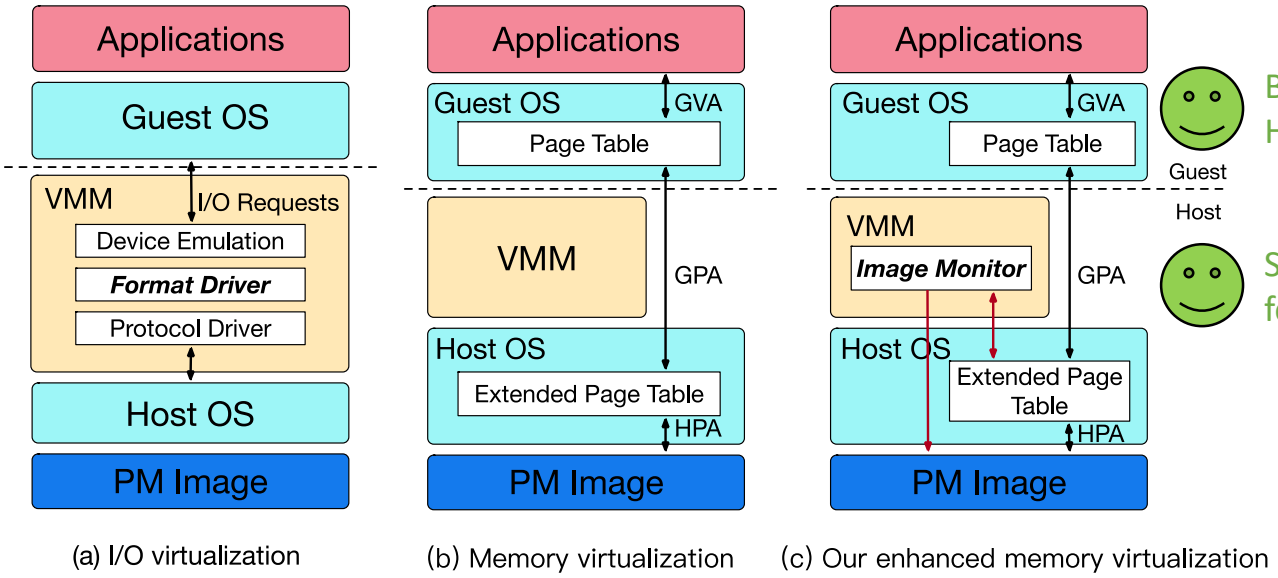

QEMU's Memory Virtualization

Byte-addressable & High Performance

Storage virtualization not implemented!

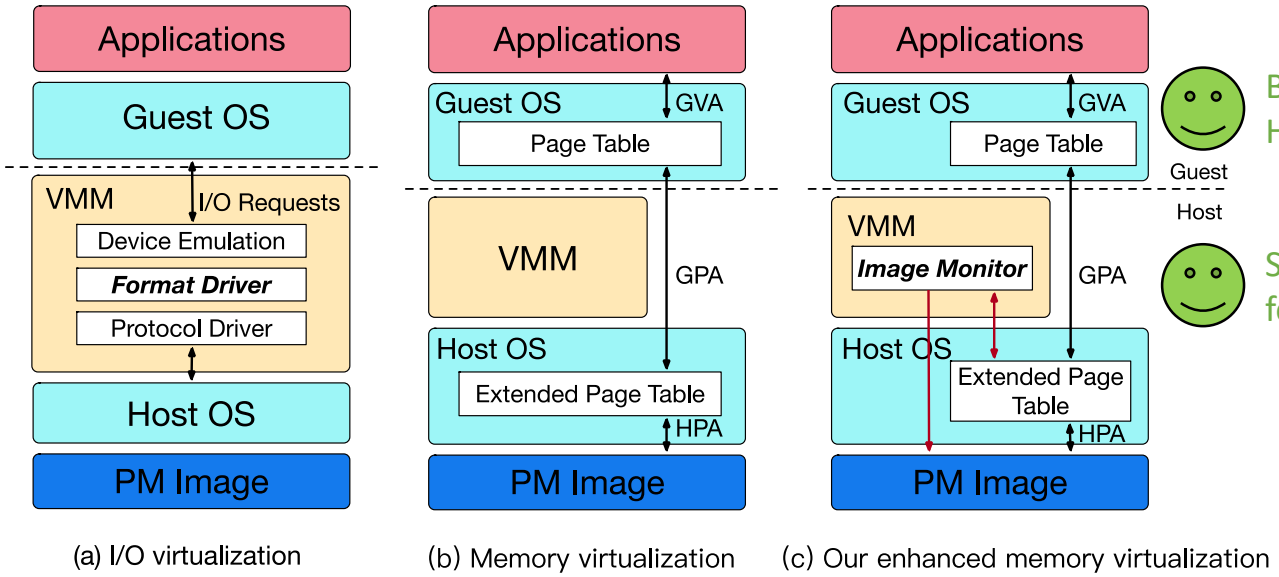
## Storage Virtualization Features

- **Thin-provision** tends to promise users a large storage space while allocating much smaller space at the beginning.
- **Snapshot** protects the data as read-only after a snapshot is taken. It provides user the option to roll-back the image to any snapshot point.
- **Base image** is also called template, it provides the opportunity to build a new image based on images created before.




(a) I/O virtualization (b) Memory virtualization (c) Our enhanced memory virtualization

| | I/O Virtualization | Memory Virtualization | Our Scheme |
|---|---|---|---|
| Byte-addressability (PM form in Guest) | ✗ | ✓ | ✓ |
| Storage Virtualization (Image management in host) | ✓ | ✗ | ✓ |




Challenge:

Data access by-pass the VMM when using memory virtualization.

Opportunity:

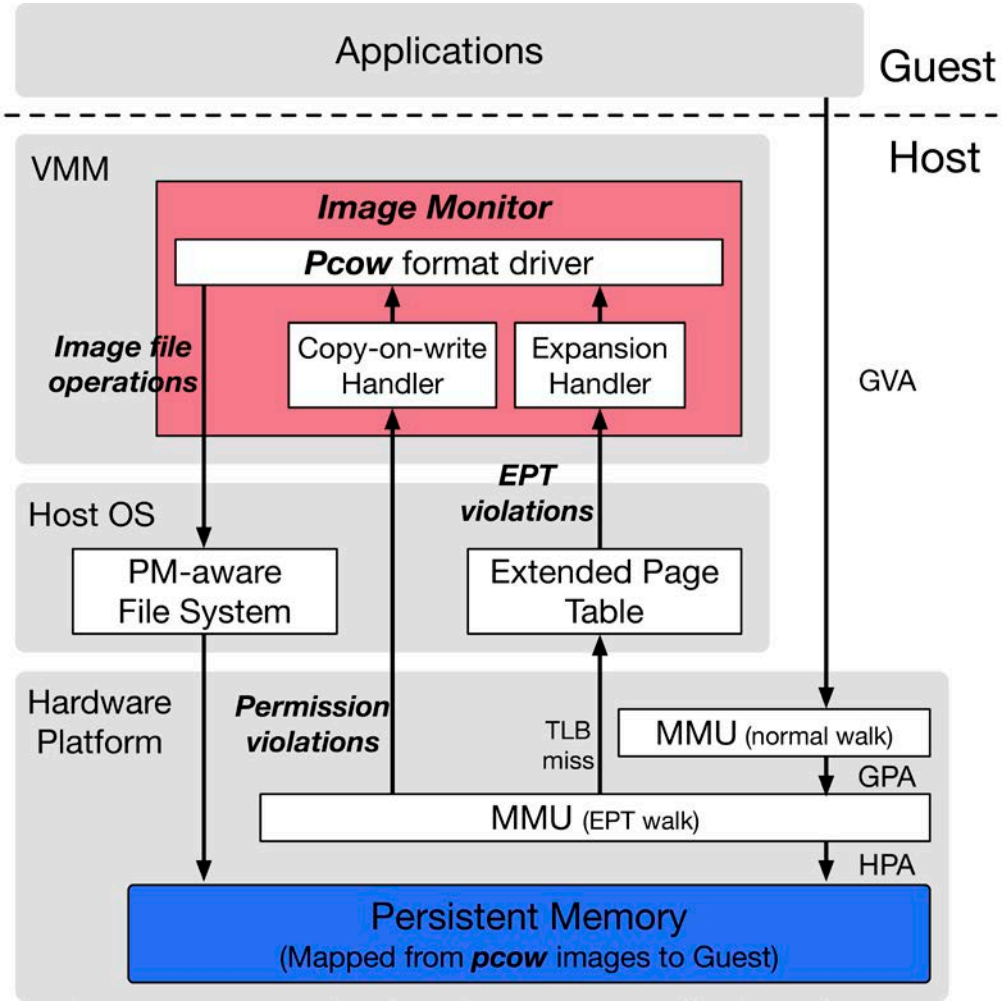
PM can take advantage of hardware-assisted address translation designed for memory virtualization (nPT or EPT) to perform the translation between virtual PM address and image file offset.

Enhance QEMU's memory virtualization mechanism by an ***Image Monitor.***

Design a VM image format called ***Pcow*** (short for PM Copy-On-Write).

Three storage virtualization features implemented with help of ***Image Monitor*** and the ***Pcow*** format:

- Thin-provision
- Snapshot
- Base image (templete)



# Agenda

- Background & Motivation
- Design & Optimization
- Performance Evaluation

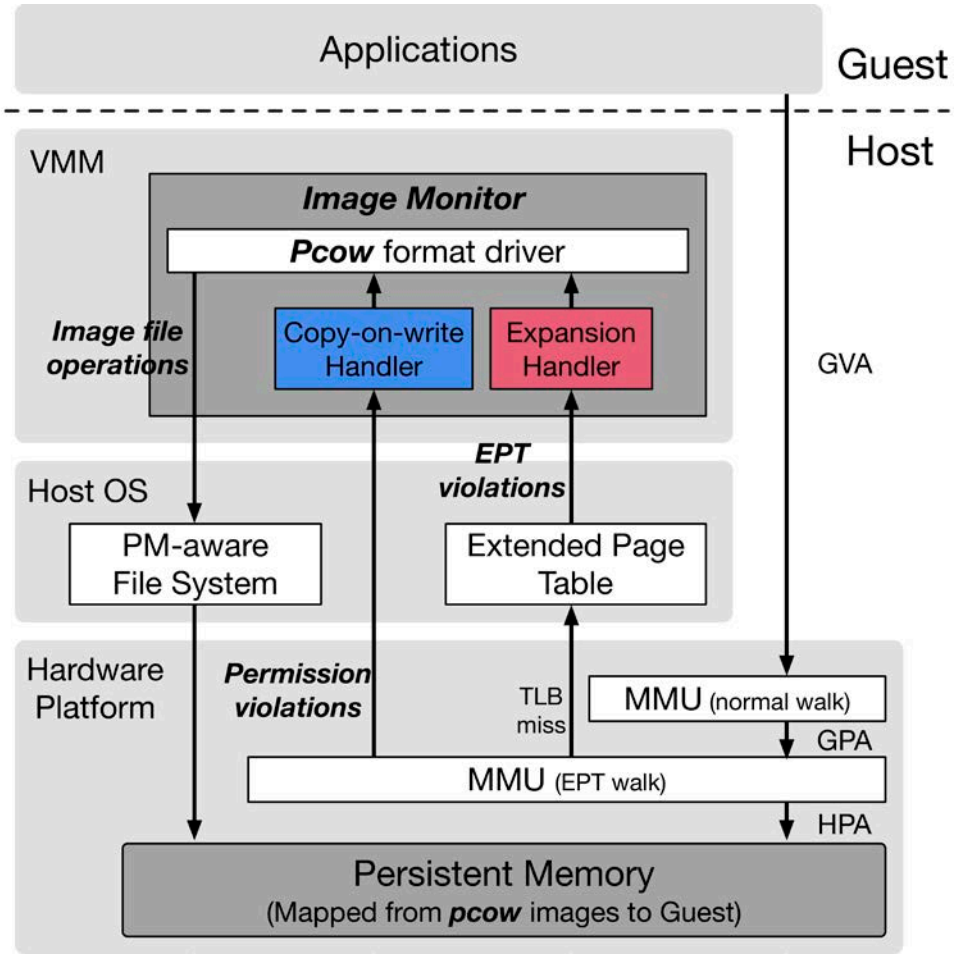
# Image Monitor | Pcow Format | Details | Optimization

## Expansion handler

- Expands the image file on demand.
- The basis of thin-provison, snapshot and base image features.
- An user-space page fault handler (Linux's new userfaultfd feature).

## Copy-on-write handler

- Protects read-only data from being written using copy-on-write.
- The basis of snapshot and base image features.
- An SIGSEGV signal handler. (Raised when writing to a write-protection area)



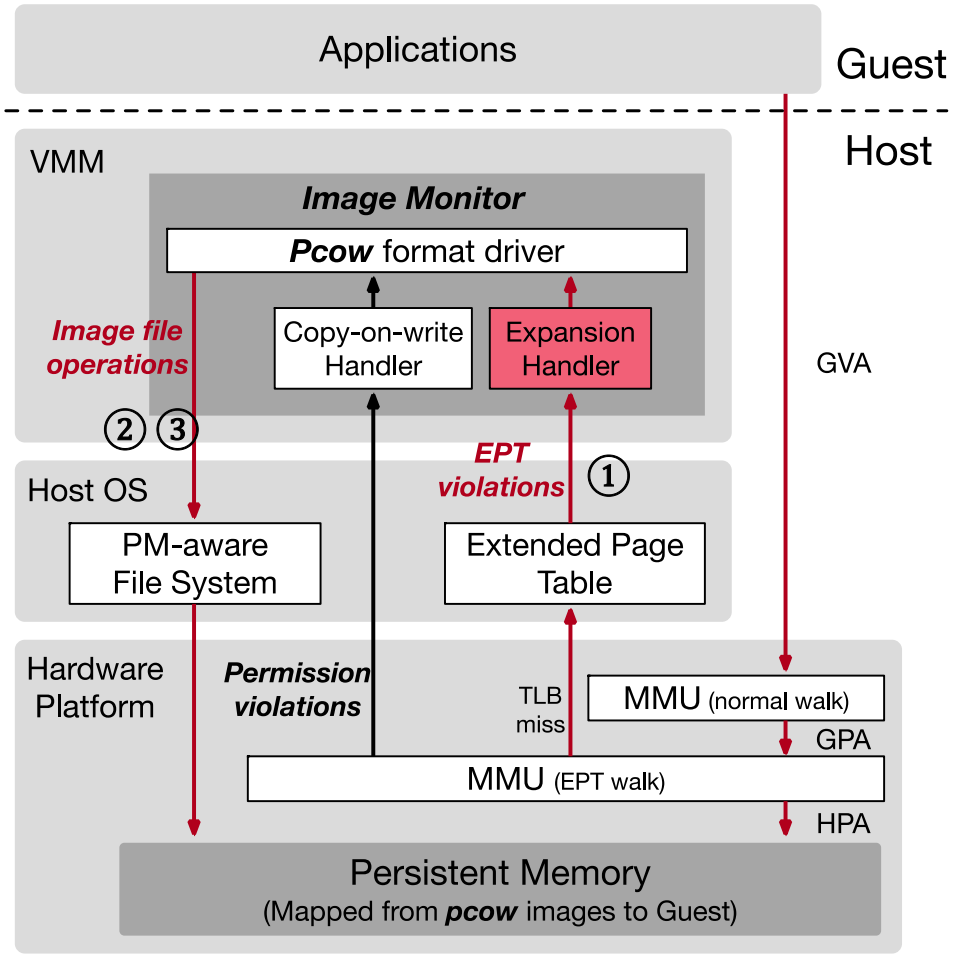
# Image Monitor | Pcow Format | Details | Optimization

## Expansion handler

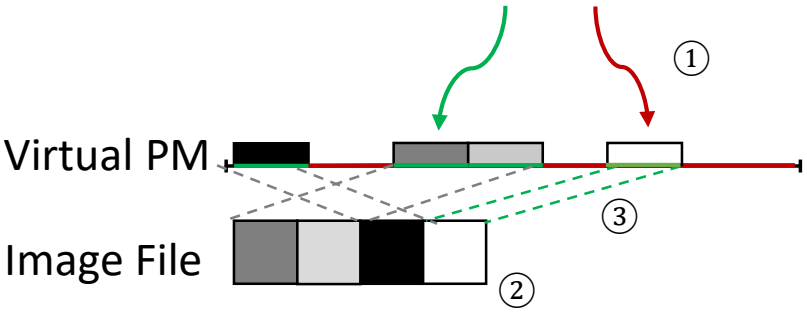
- Expands the image file on demand.
- The basis of thin-provison, snapshot and base image features.
- An user-space page fault handler (Linux's new userfaultfd feature).

## Copy-on-write handler

- Protects read-only data from being written using copy-on-write.
- The basis of snapshot and base image features.
- An SIGSEGV signal handler. (Raised when writing to a write-protection area)



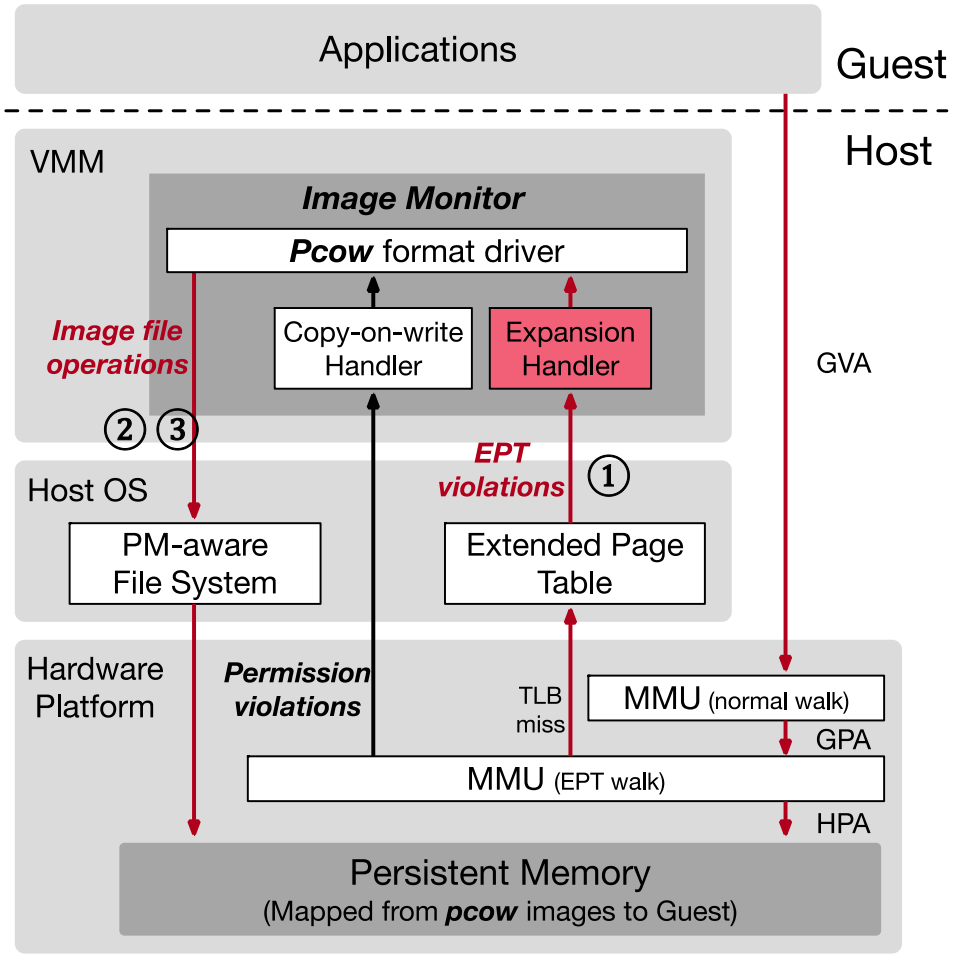
## Expansion handler




① Guest Apps touch a page with no PM image file backed, the Expansion Handler is invoked.

② Pcow format driver allocates a new block at the end of the pcow image file.

③ Expansion Handler maps the newly allocated block to the fault address.



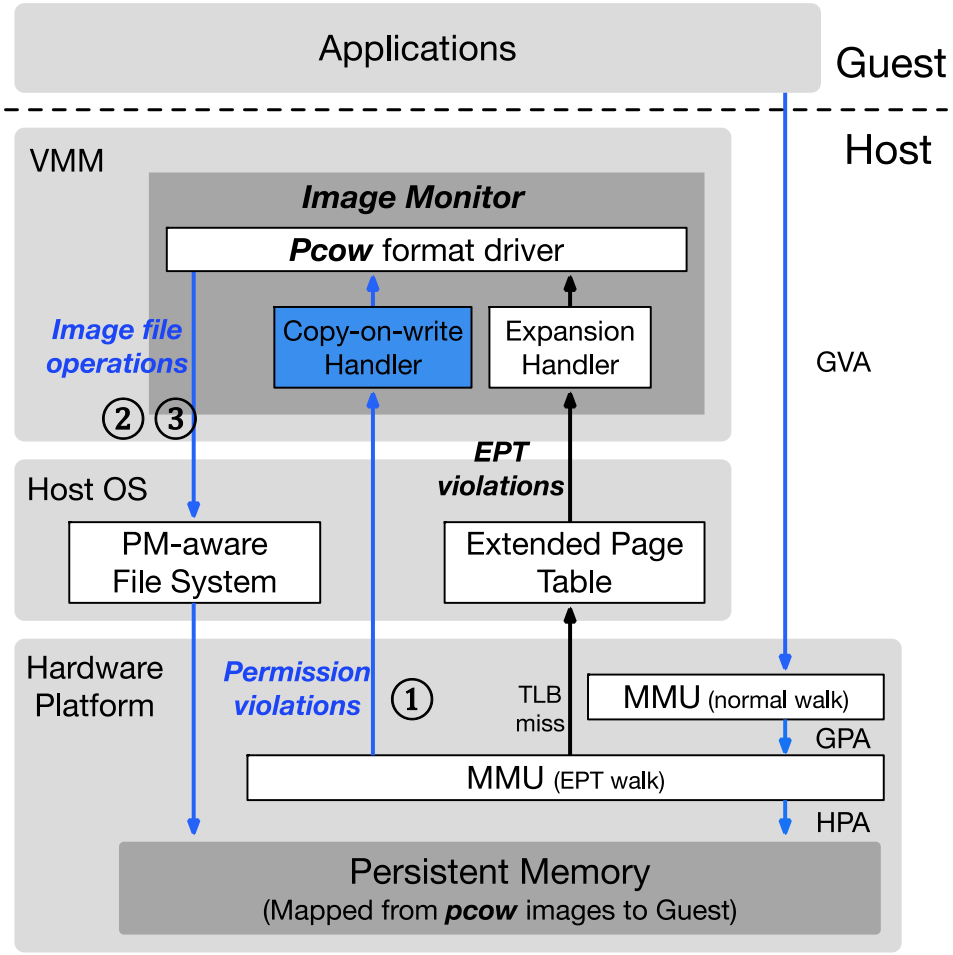
# Image Monitor | Pcow Format | Details | Optimization

## Expansion handler

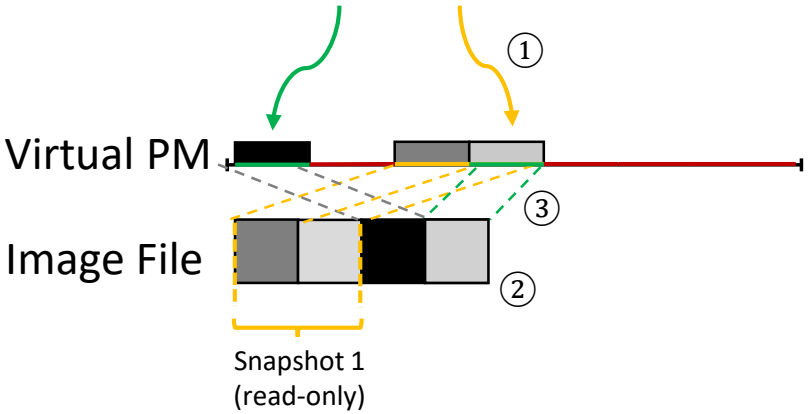
- Expands the image file on demand.
- The basis of thin-provison, snapshot and base image features.
- An user-space page fault handler (Linux's new userfaultfd feature).

## Copy-on-write handler

- Protects read-only data from being written using copy-on-write.
- The basis of snapshot and base image features.
- An SIGSEGV signal handler. (Raised when writing to a write-protection area)



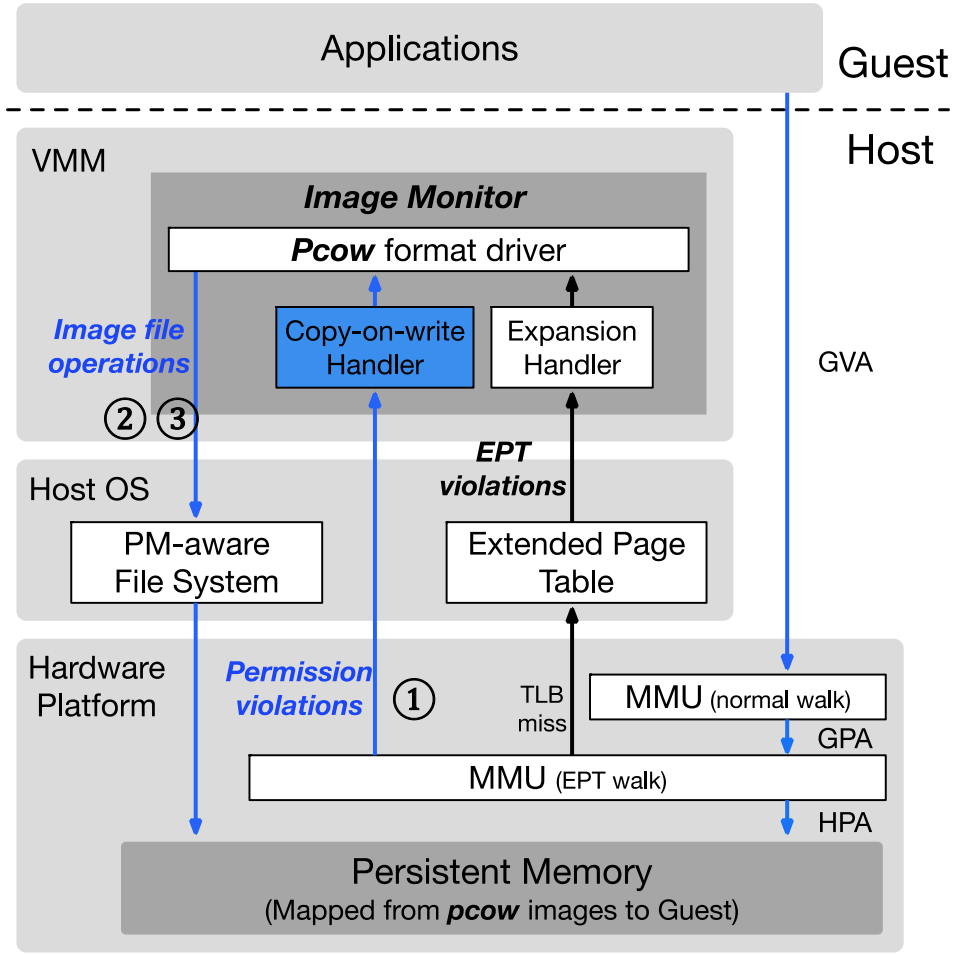
## Copy-on-write Handler




① Guest Apps access an read-only page, the Copy-on-write Handler is invoked.

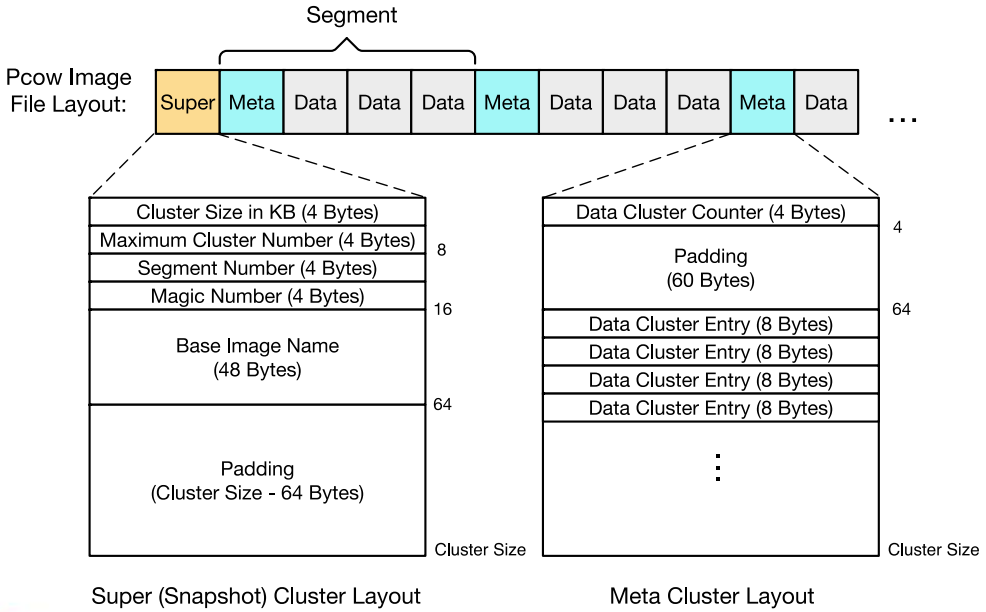
② Pcow format driver allocates a new block at the end of the image file and do COW.

③ Copy-on-write Handler maps the COWed block to the write permission violation address.



## Pcow Image File Layout

- Data and meta-data is organized in fixed-size clusters.
- New clusters are created in an appending manner.
- Much more concise compared with IO virtualization formats like qcow2.



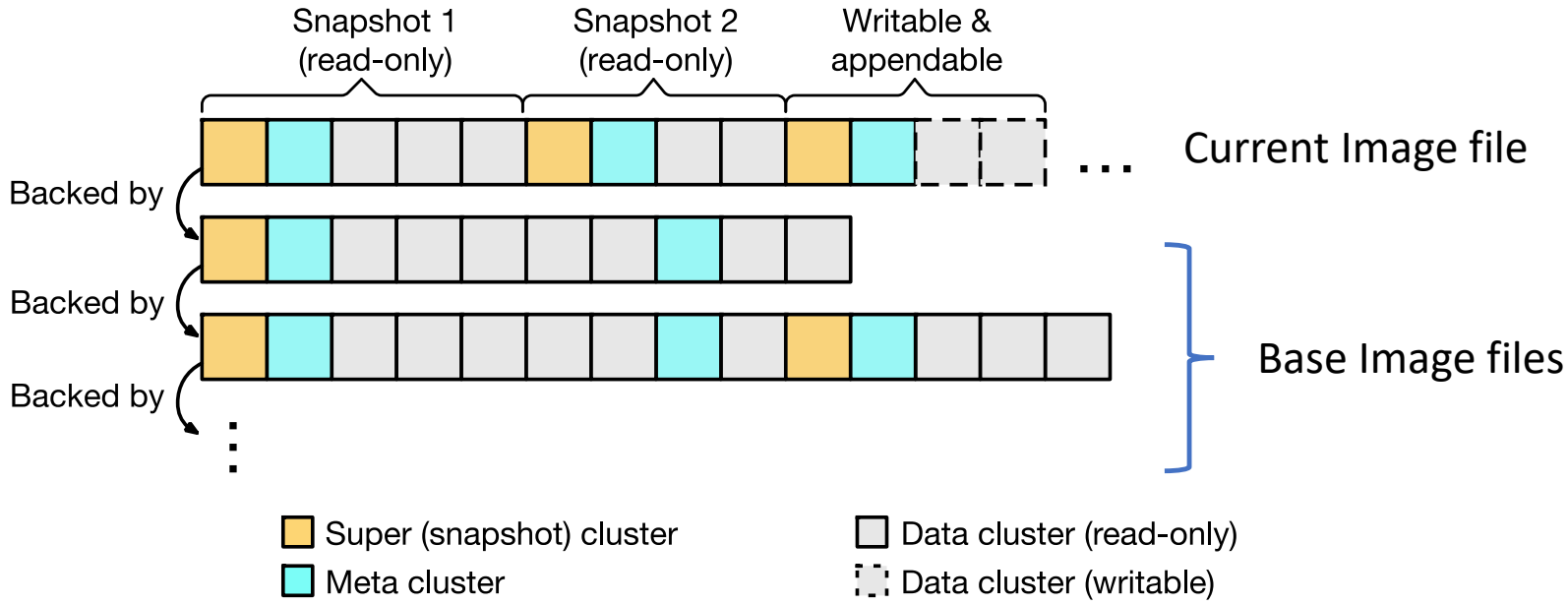
- Necessary clflush and sfence instructions are used to maintain for the crash consistency of meta-data.

```
data_cluster_num ← (virt_addr − base_addr)/
clflush(&data_cluster_num, cacheline_size)
sfence()
cluster_counter ← cluster_counter + 1
clflush(&cluster_counter, cacheline_size)
sfence()
cluster_off ← file_len + cluster_size
```

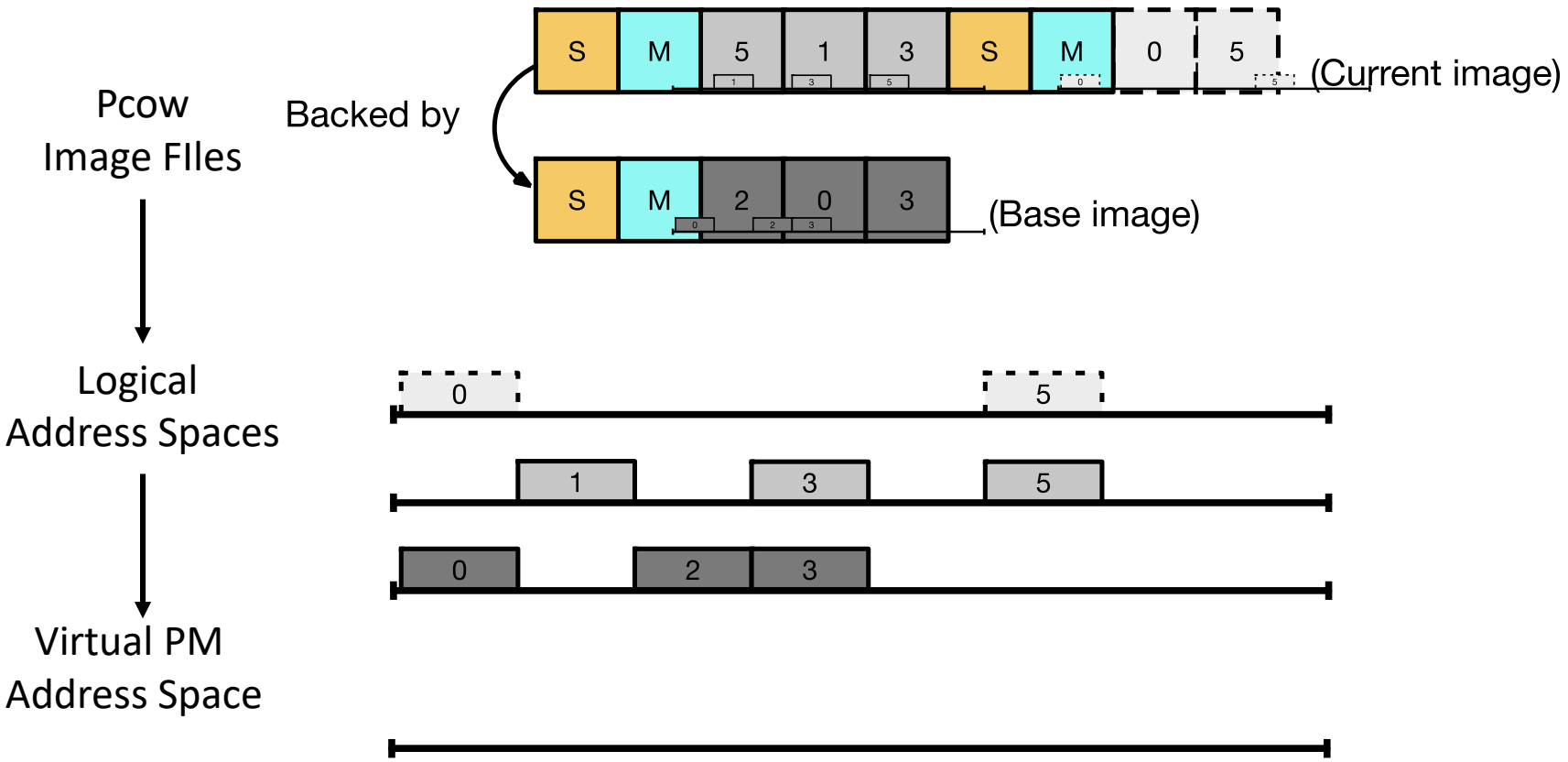
- Some meta-data that needs to be updated frequently is stored in one cacheline size.

## A Pcow Image Example

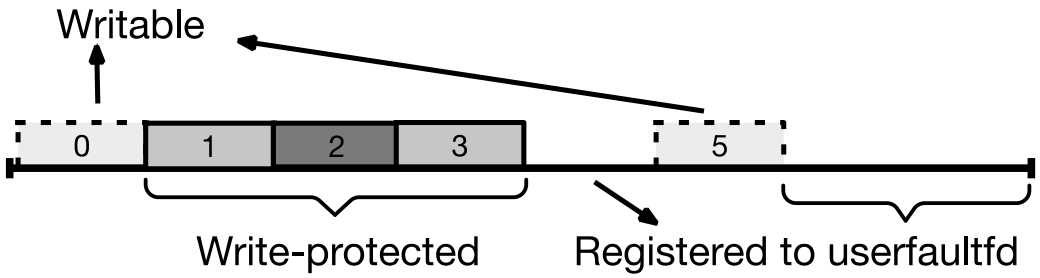
- **Thin-provision:** The image file is very much when created.
- **Base image:** A current image file is created based on the 2 base image file.
- **Snapshot:** The current image file consists of 2 snapshot part a writable part



## Pcow Mapping at Startup



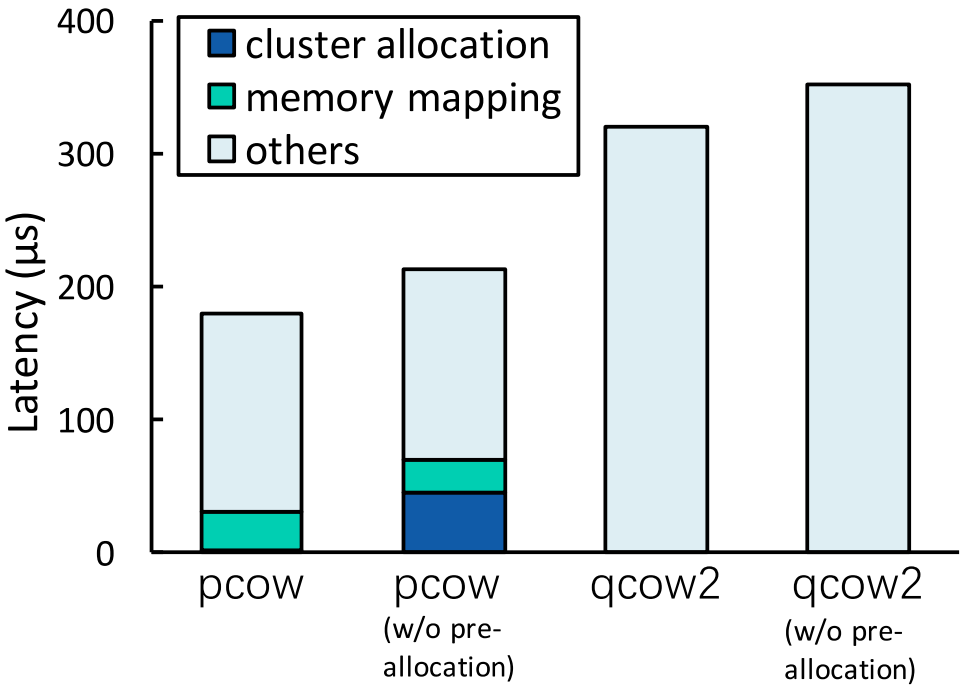
## Pcow Updating at Runtime




- Writeable area can be read or write by the Guest Apps.
- Write to the write-protected area will invoke the Copy-on-write Handler.
- Read / write the userfaultfd area will invoke the Expansion Handler.
- Copy-on-write Handler and Expansion Handler do image file operations and update the EPT page table.

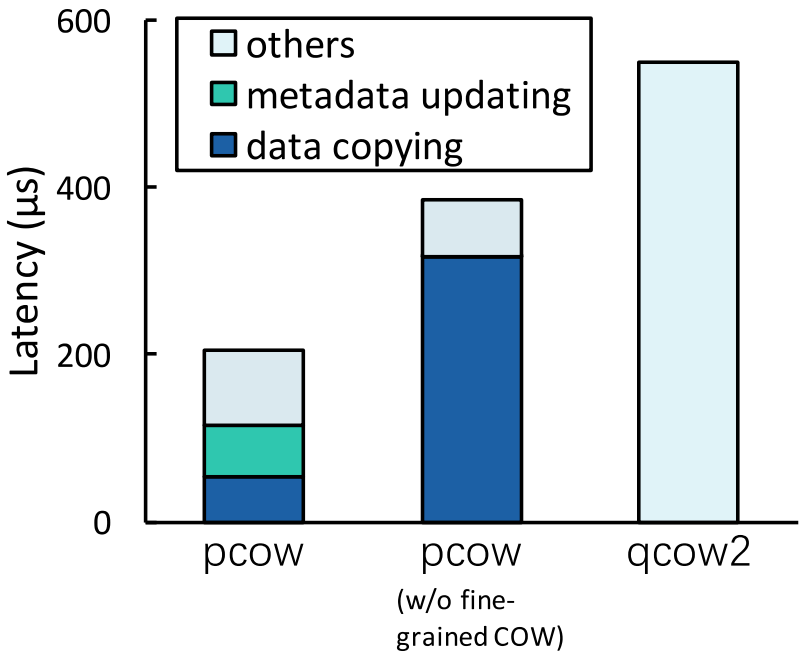
## Pre-allocation

- Dedicated cluster allocation thread is use for cluster pre-allocation.
- Decreases the image expansion latency by 45 us.



## Fine-grained Copy-on-write

- Copy 4KB instead of 64KB cluster size for lower COW latency.
- Decreases the copy-on-write latency by about 200 us.




# Agenda

- Background & Motivation
- Design & Optimization
- Performance Evaluation

Prototype implemented based on QEMU 3.0.

Our physical PM device is emulated by a DRAM partition.

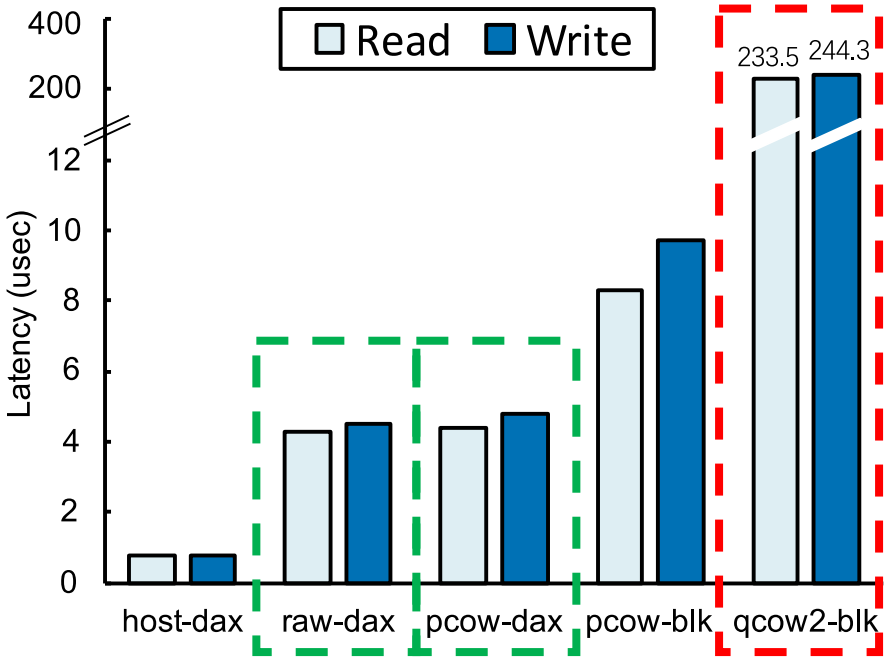
Comparisons between:

- Our prototype (pcow)
- Native memory virtualization (raw)
- I/O virtualization image format (qcow2)

| | |
|---|---|
| **CPU** | Intel Xeon E5-2609 1.70 GHz ×2 |
| **CPU cores** | 8 ×2 |
| **Processor cache** | 32 KB L1i, 32 KB L1d, 256 KB L2, 20 MB L3 |
| **DRAM** | 80 GB |
| **PM** | 48 GB (emulated by DRAM) |
| **OS** | CentOS 7.0, kernel version 4.16.0 (same in VMs) |
| **VMM** | QEMU version 3.0.0 |
| **File system** | ext4 (mounted with DAX option) |

## Pcow-dax:

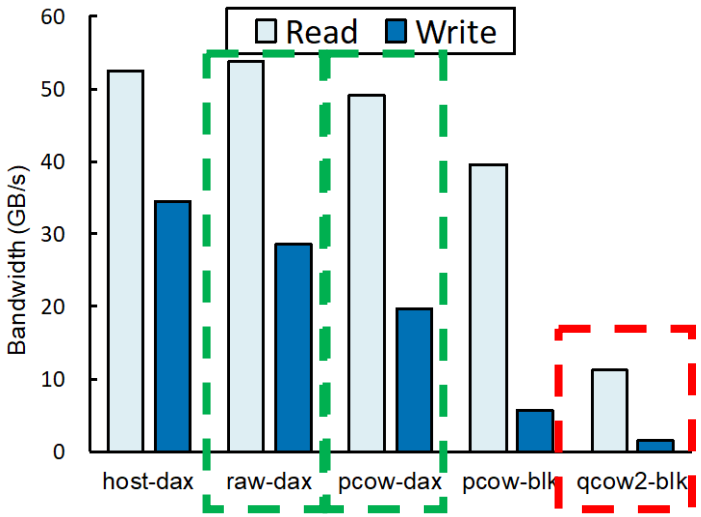
- No overhead compared with native memory virtualization (raw-dax).
- About 50x better than qcow2-blk.

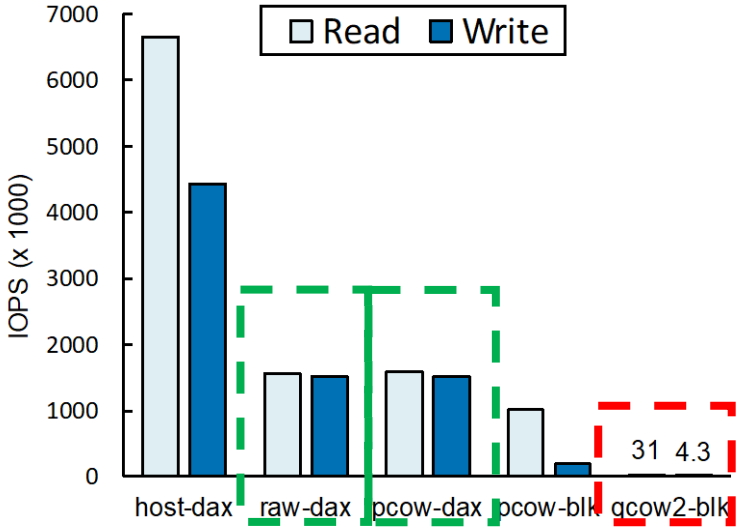



- Fio 4KB single thread
- -dax: mmap interface
- -blk: read / write interface

## Pcow-dax:

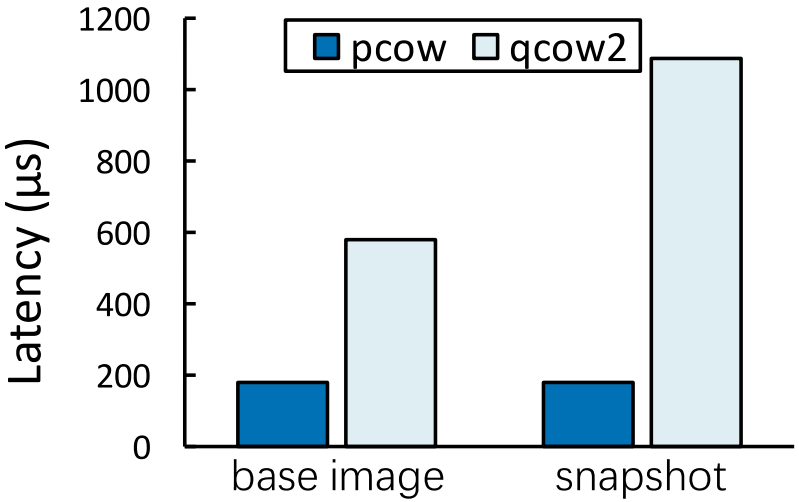
- No overhead compared with native memory virtualization (raw-dax).
- Bandwidth 4x better than qcow2, IOPS hundreds of times better than qcow2.

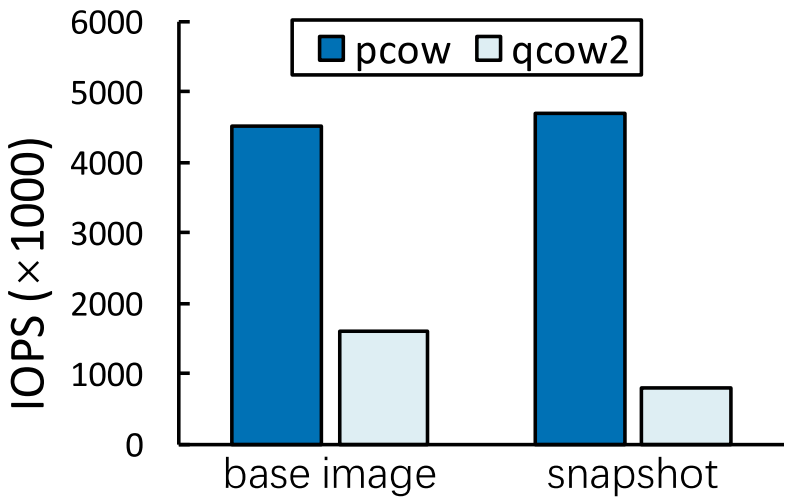


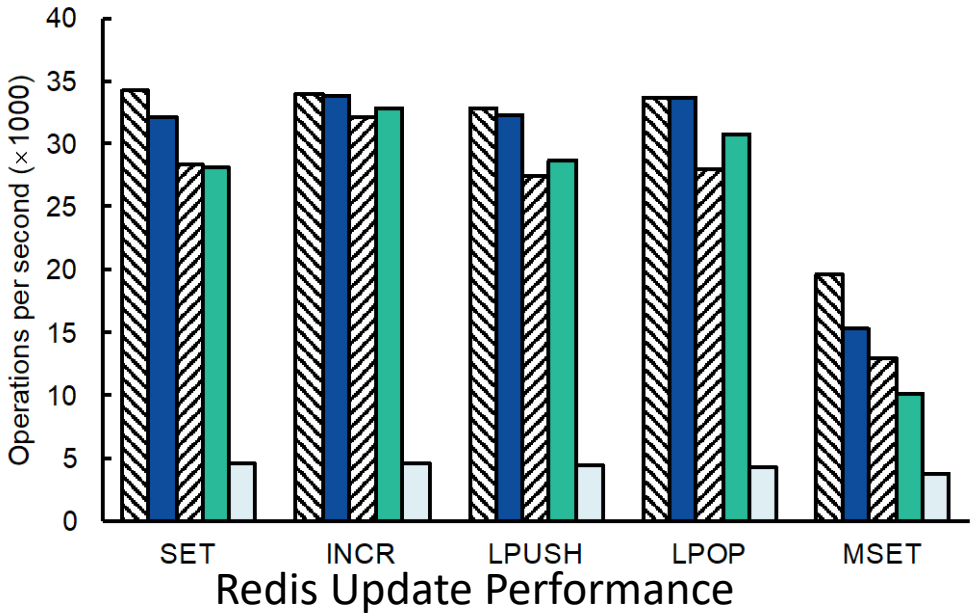



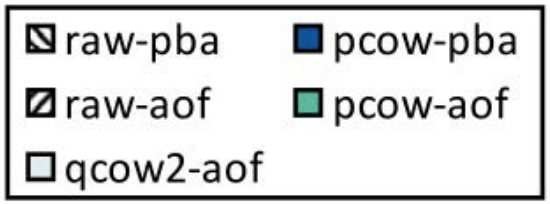

- Bandwidth: Fio 1MB 16threads
- IOPS: Fio 4KB 16threads
- -dax: mmap interface
- -blk: read / write interface

- Pcow's copy-on-write performance is about 3x better than qcow2.






- Native memory virtualization (raw-)
- Our scheme (pcow-)
- I/O virtualization image format (qcow2-)

- Redis (-aof)
- Redis-PMDK (-pba)

- Redis-PMDK (pcow-pba) still have better performance than Redis (pcow-aof) when using our scheme.
- Our scheme is still compatible with the real-world application's optimization for PM in virtual machines.

# Summary

- We achieve both virtual PM byte-addressability and image management.
- We implement 3 storage virtualization features for virtual PM.
- We take advantages of EPT for address translation between virtual PM and pcow image file offset.
- Our scheme is up to 50x faster than I/O virtualization image format qcow2. Almost no overhead compared with the native memory virtualization.

## Source Code Released:

https://github.com/zhangjaycee/qemu-pcow

## Usage:

**Pcow manage tool "pcow-img":**

```
pcow-img  create  64(KB)  128(GB)  my_pcow_file.img
```

**QEMU parameters:**

```
qemu-sysmte-x86_64 ... \
-object memory-backend-file,id=pm,mem-path=my_pcow_file.img,format=pcow,share=on,discard-data=off,merge=off \
-device nvdimm,id=pm,memdev=pm \
...
```

# Thanks! Questions?