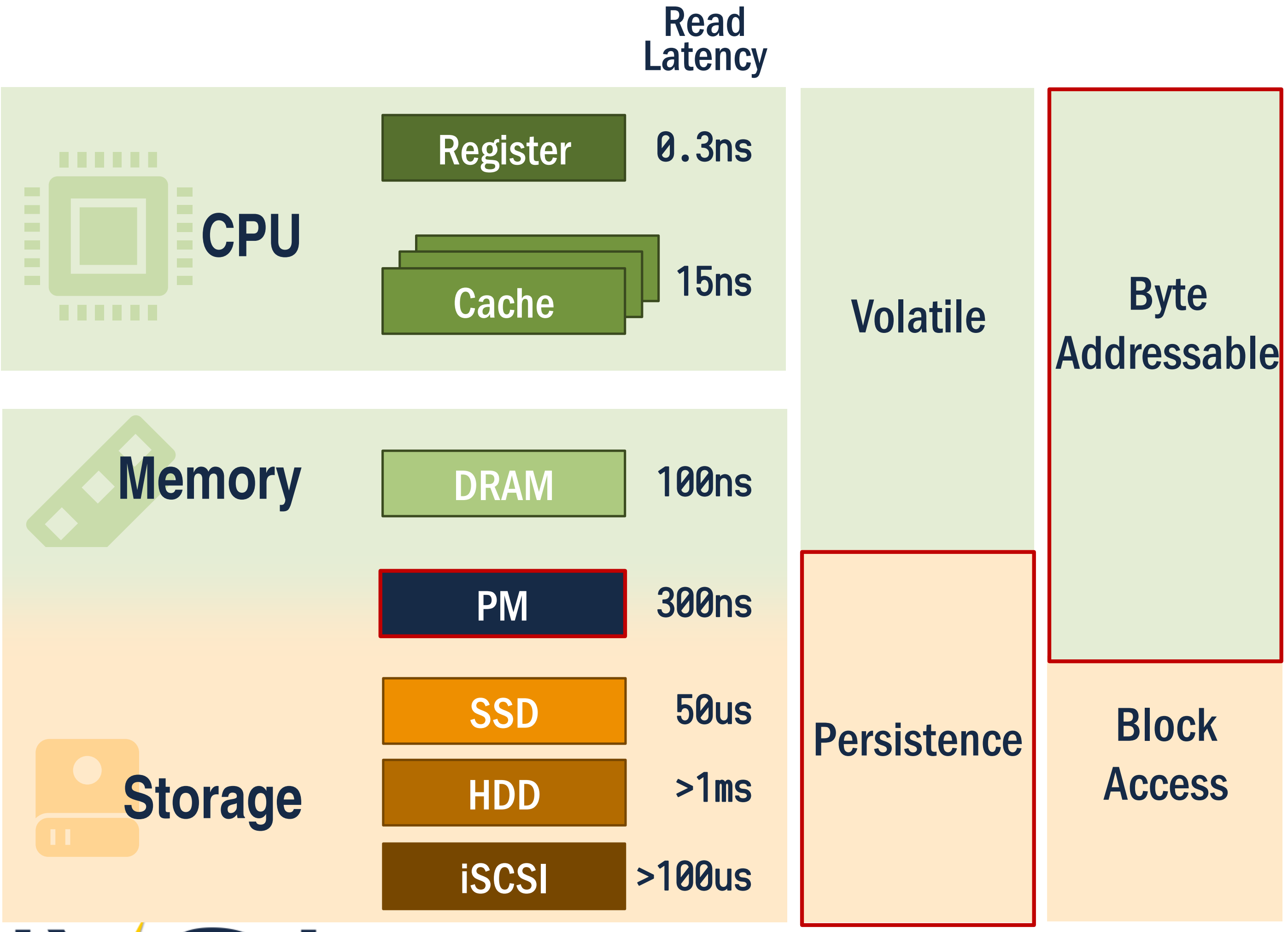
FileMR: Rethinking RDMA Networking for Scalable Persistent Memory

Jian Yang *(UC San Diego)*

Joseph Izraelevitz *(University of Colorado, Boulder)*

Steven Swanson *(UC San Diego)*

Persistent memory and RDMA



		Read Latency
CPU	Register	0.3ns
	Cache	15ns
Memory	DRAM	100ns
	PM	300ns
Storage	SSD	50us
	HDD	>1ms
	iSCSI	>100us

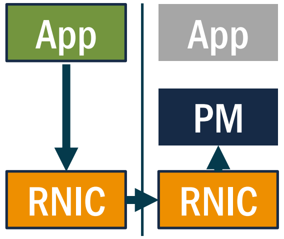
Volatile

Persistence

Byte Addressable

Block Access

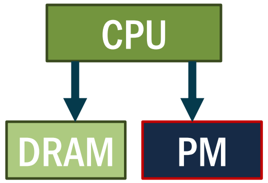
Remote Direct Memory Access (RDMA)



- DMA access across nodes
- One-sided (read/write)

Fast, fine-grained remote access

Persistent Memory (PM, aka. SCM, NVMM)



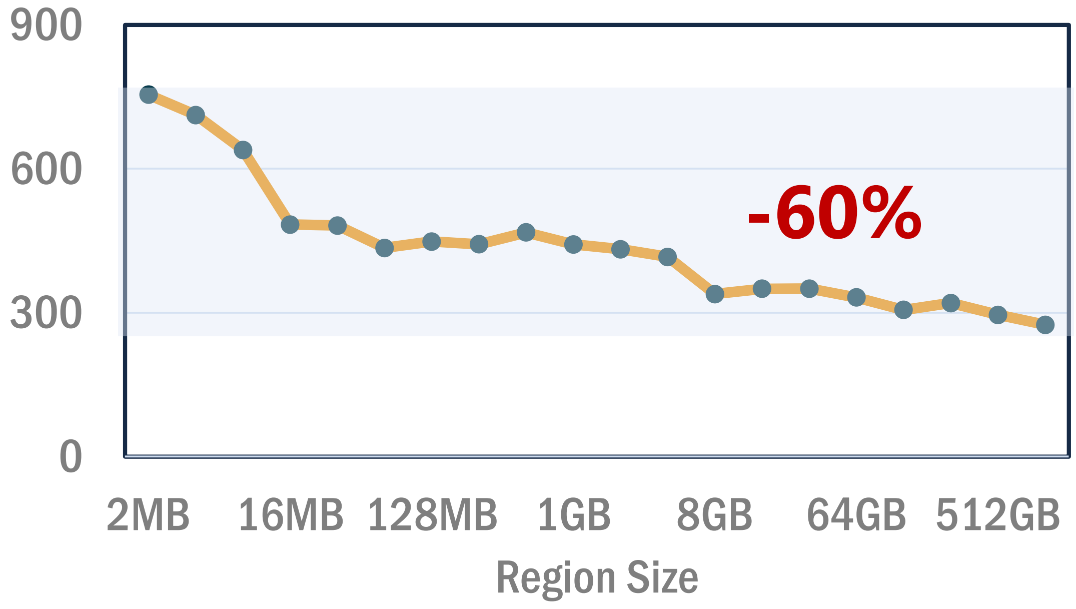
- PCM, STT-RAM, 3DXPoint
- Co-locate with DRAM
- Higher density than DRAM
- Byte-addressable: load/store

Fast, fine-grained persistence

Persistent memory and RDMA

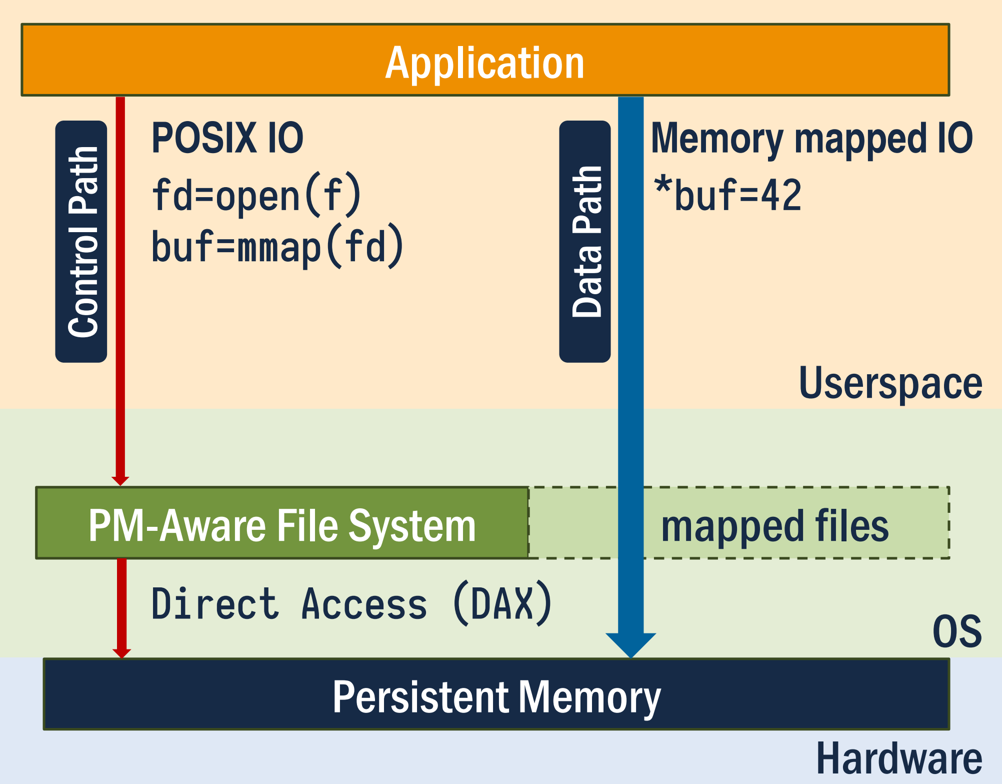
- RDMA on PM != Fast, fine-grained persistent remote access

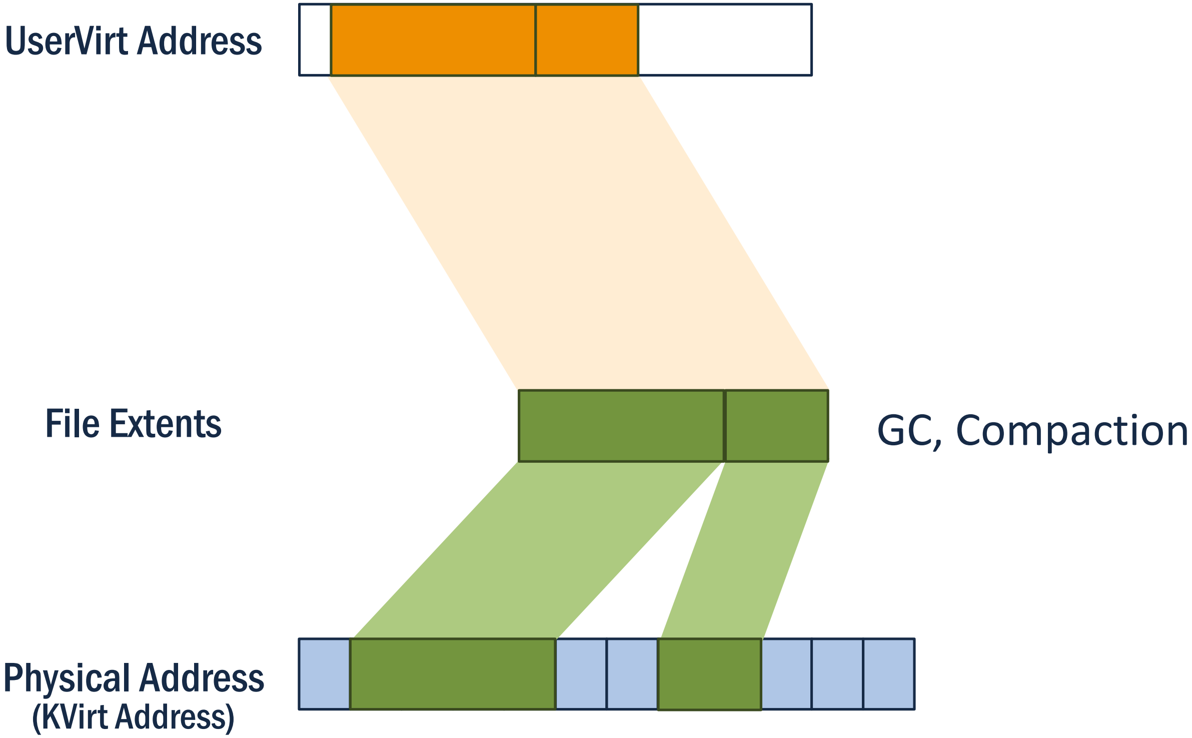
Throughput (kiops) of random 4KB RDMA writes on PM (Optane)



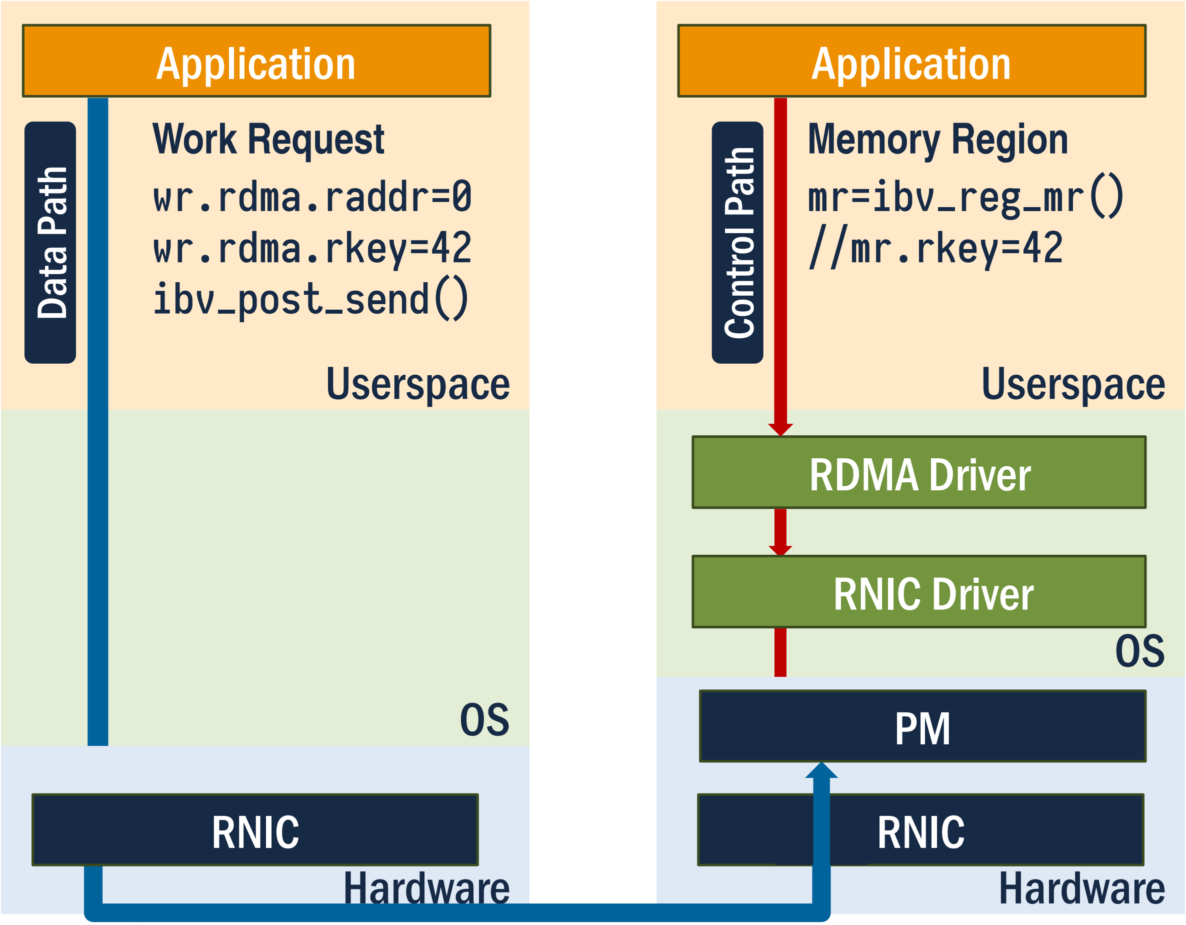
- Other issues: allocations, protection, naming …

PM: memory management





RDMA memory management

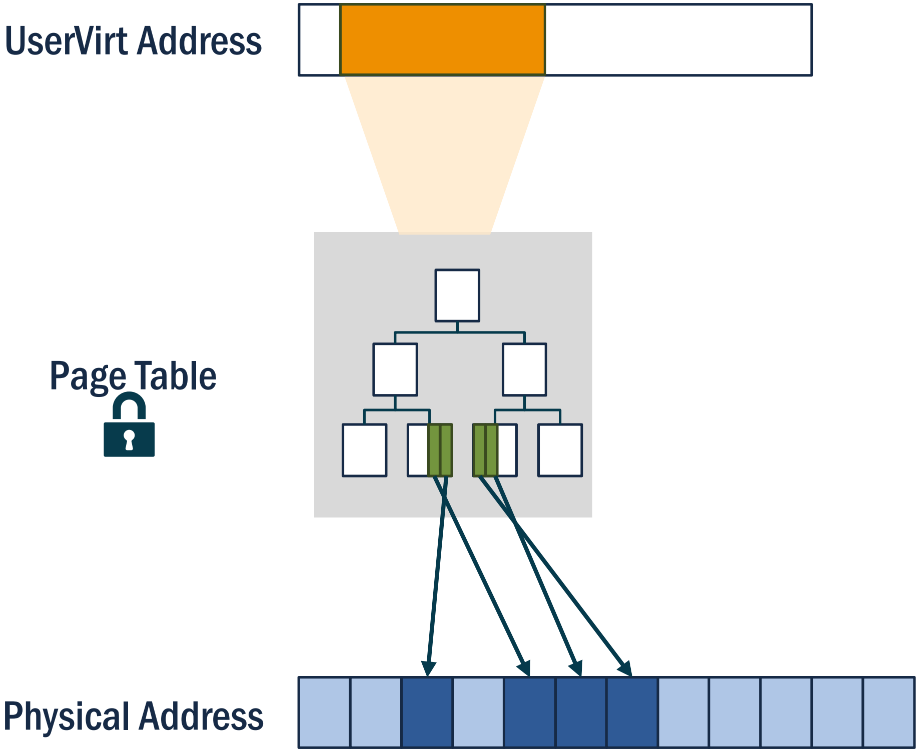


Work Request

```
wr.rdma.raddr=0
wr.rdma.rkey=42
ibv_post_send()
```

Memory Region

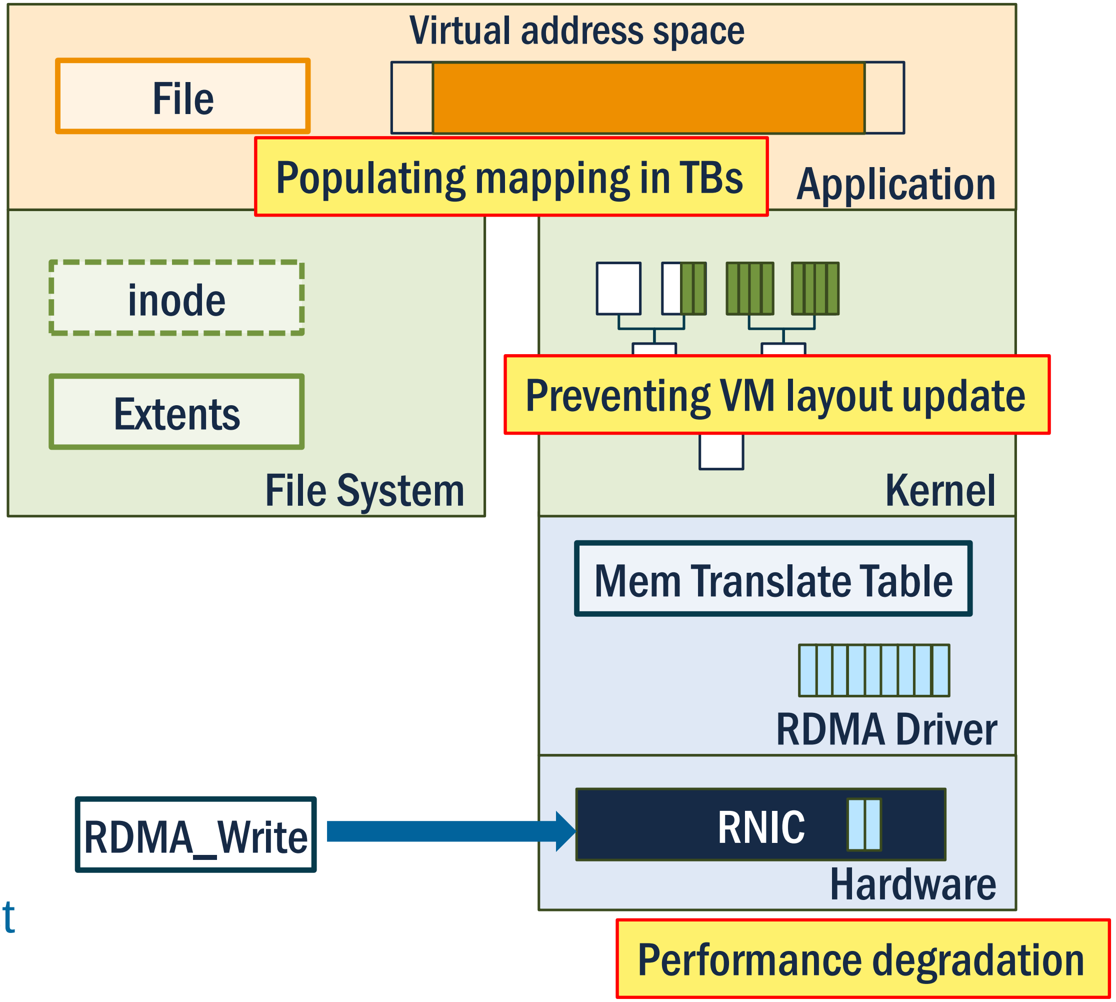
```
mr=ibv_reg_mr()
//mr.rkey=42
```



RDMA on PM

```
fd = open(/mnt/file)
p = mmap(…, fd, …)
# RDMA connection setup
ibv_reg_mr(pd, p, …)
# incoming writes from a remote node
< ibv_post_send(…, wr{RDMA_WRITE, addr,…})
```

- Virtual memory related issues:
  - Linux VM subsystem was not designed for the usage of PM, optimizations require remapping
  - RDMA is not designed for large, long-term memory
- Issues discussed in paper:
  - Security, naming, isolation, connection management, replication, persistence, multicast



Populating mapping in TBs

Preventing VM layout update

Performance degradation

Alternative approaches to VM issues on RDMA

Holistic Design:

- Co-design PM management software and RDMA networking
- VM mitigation: PUD(1GB) pages, data structure
- E.g., PASTE, Mojim, Hotpot, LITE, librpmem

  NSDI 18 ASPLOS 15 SOCC 15 SOSP 17
- Require dedicate APIs and application redesign

Indirection:

- Existing interfaces (e.g. POSIX IO) as indirection
- E.g., Octopus, Orion, RDMA key-value stores

  ATC 17 FAST 19
- No userspace direct access

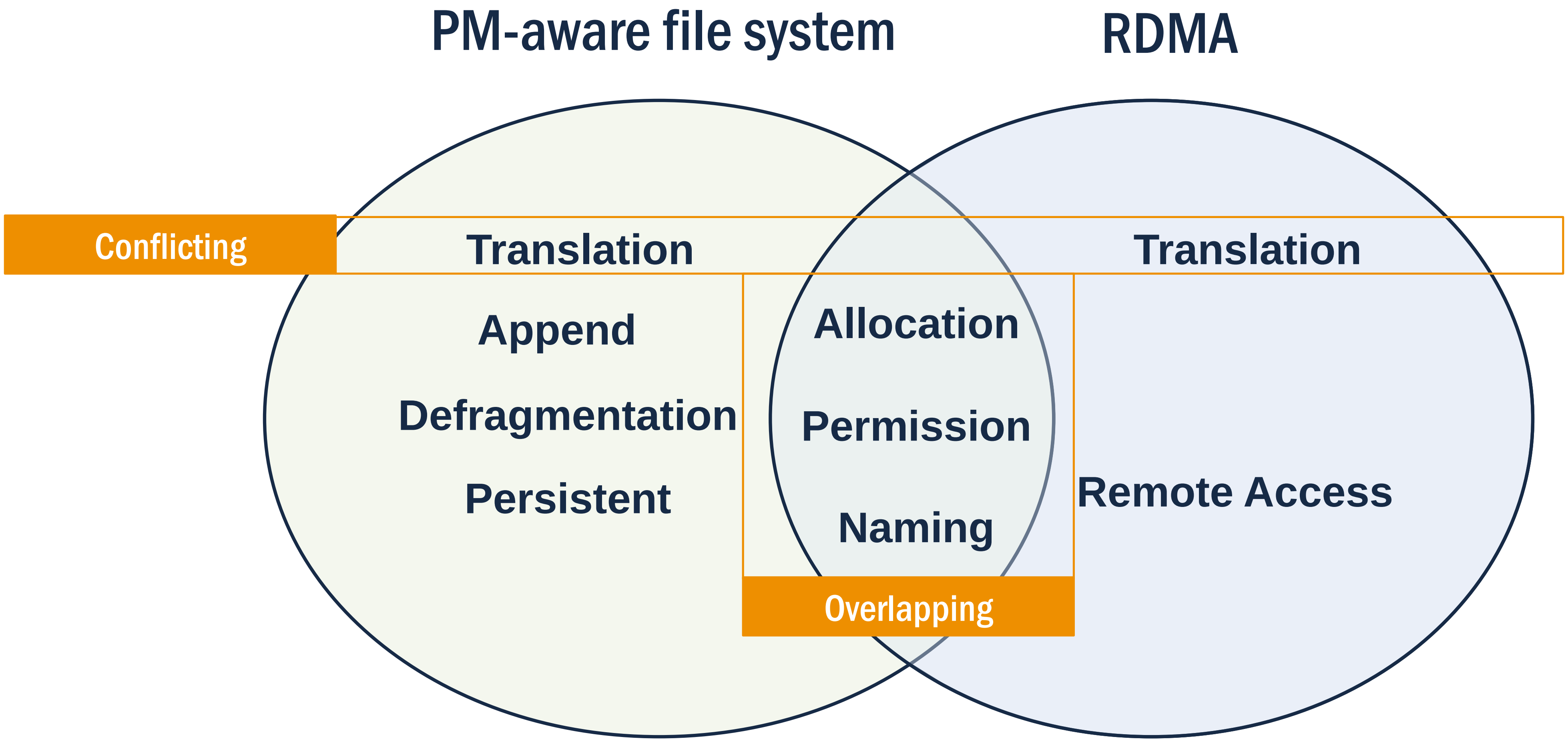
NIC Pagefault:

- Using on-demand paging (IO page faults)
- MR registration is O(1)
- #IOPF is expensive (300+µs on mlx5)

Physical address on wire:

- No translation needed
- MR registration is O(1), no translation
- Security issues

Conflicting roles of metadata management



PM-aware file system

RDMA

Conflicting

Translation

Translation

Append

Defragmentation

Persistent

Allocation

Permission

Naming

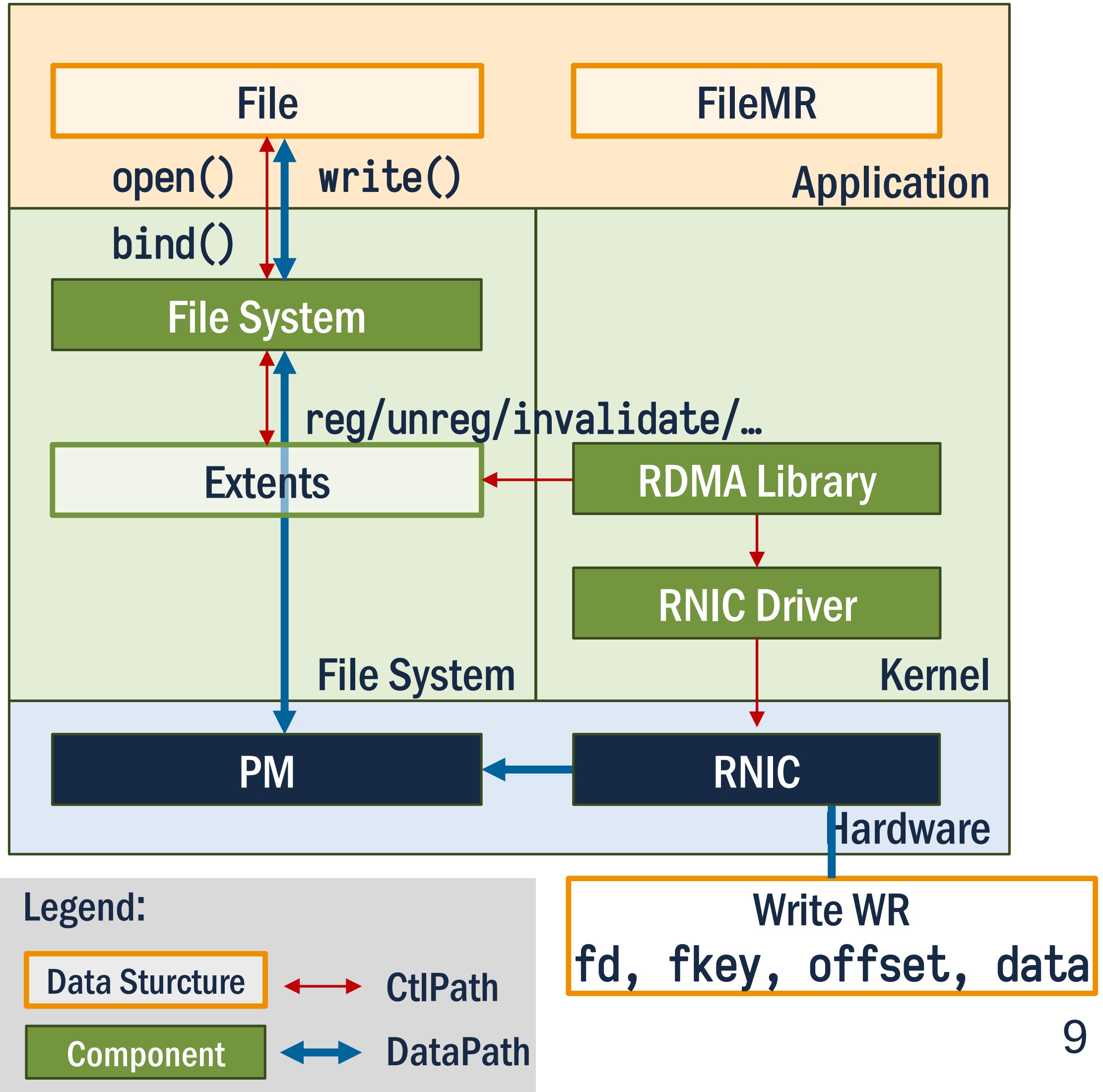
Overlapping

Remote Access

FileMR: File-based memory region

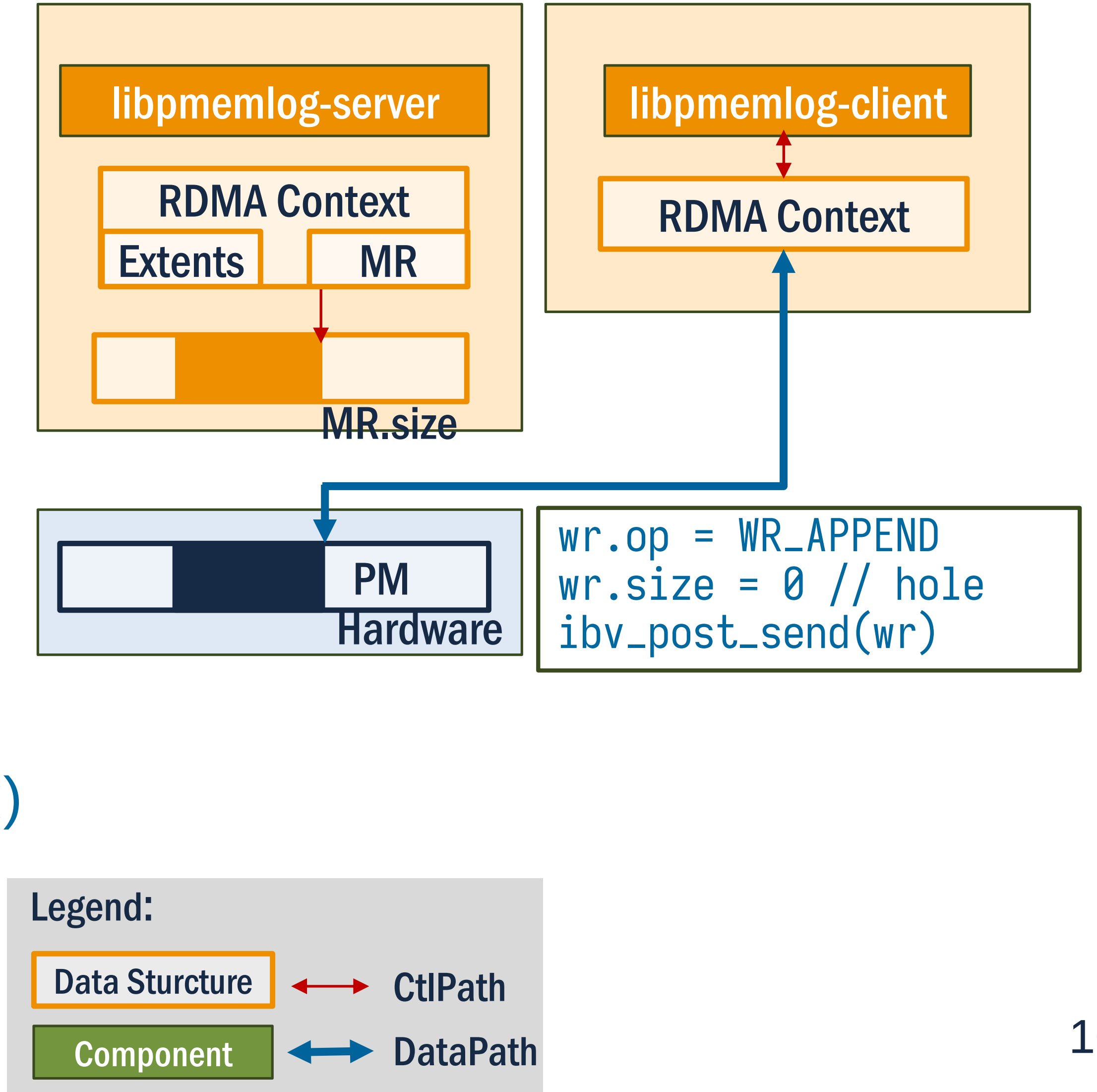
- **File-based memory region**
  - New type of memory region: FileMR
  - File system/PM Library maintains the metadata of the MR
  - File system initiates translation update
    - Decoupled with VM (file offset)
    - FS-managed protection / naming

```
fd = open(/mnt/file)
# RDMA connection setup
mr=ibv_reg_mr(pd, NULL, FILEMR)
ioctl(fd, FILEMR_BIND, mr.key, …)
# incoming writes from a remote node
< ibv_post_send(…,wr{RDMA_WRITE,offset,…})
```



Range-based memory translation table

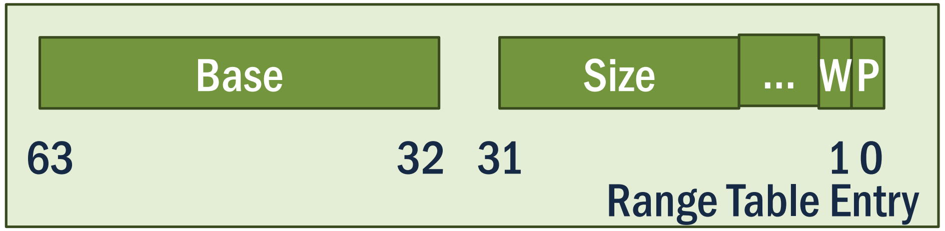
- RDMA Append
  - `APPEND` verb (write at `MR.size`)
  - Pre-provision / IO pagefault
- “File system” is loosely defined
  - Implements functions and callbacks
- **Case study: `libpmemlog`**
  - `libpmem` manages extents in userspace
  - Bypass kernel-space file system (`devdax`)

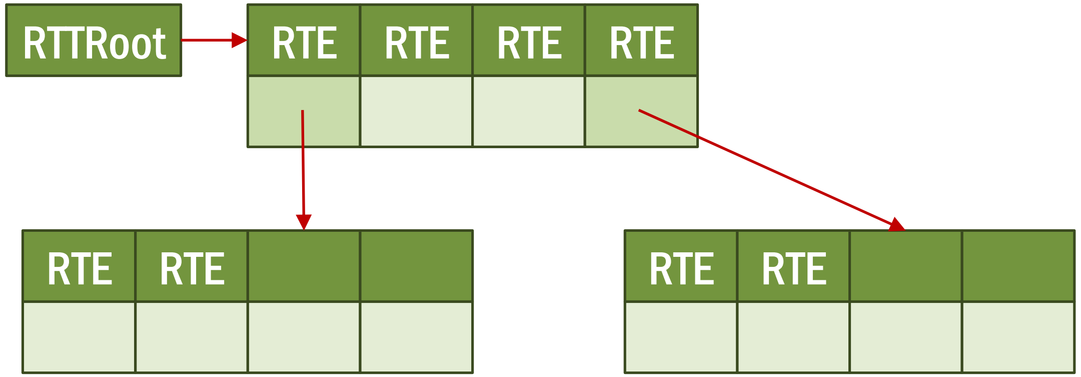


```
wr.op = WR_APPEND
wr.size = 0 // hole
ibv_post_send(wr)
```

Range-based memory translation table

- **RangeMTT: Range-based address translation**
  - Based on the design of range-based TLB[1]
  - Reverse translation between file offset to physical address
  - 4KB page-aligned, 32-bit addressing (16PB)
  - B-tree structure





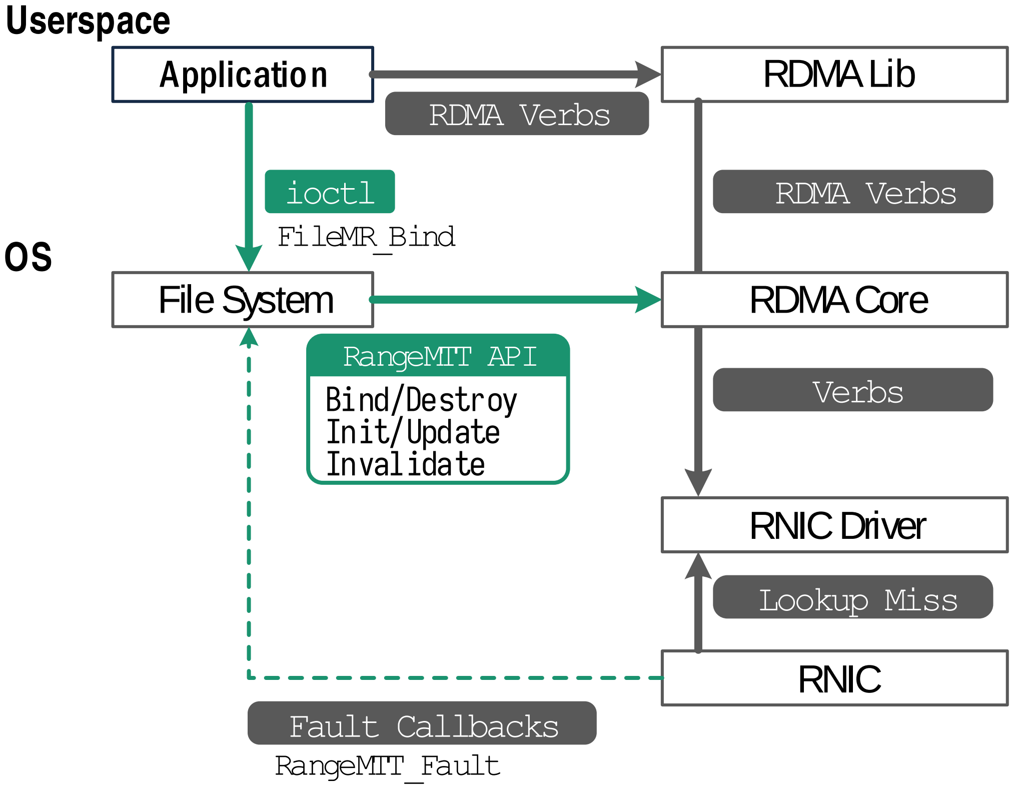
[1] Karakostas *et al*, Redundant Memory Mappings for Fast Access to Large Memories (ISCA 2015)

FileMR : Implementation

- Implement FileMR and RangeMTT on SoftRoCE (`rxe`)
  - SoftRoCE is a software RNIC based on UDP
  - Minor change throughout RDMA stack
  - Added Filesystem RangeMTT API
  - Using `ioctl` for bind API

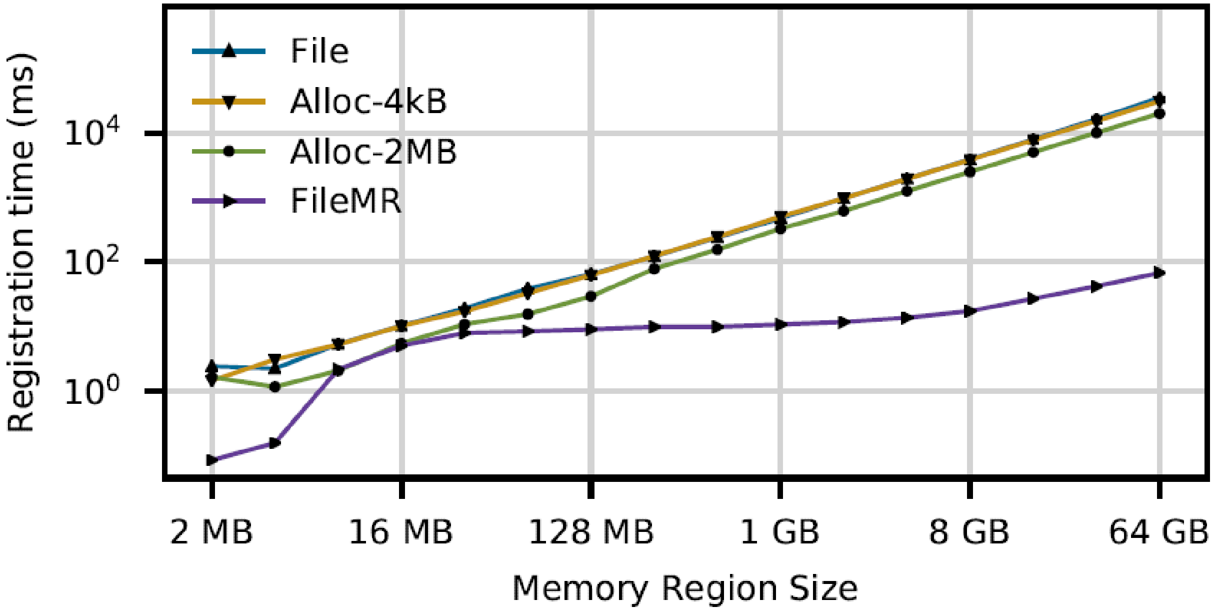
- RNIC cache emulation
  - Emulate MTT/RangeMTT cache on `rxe`
  - 4096-entry 4-way set associative

- Limitations:
  - No application-level end-to-end performance
  - Higher latency than real RNICs

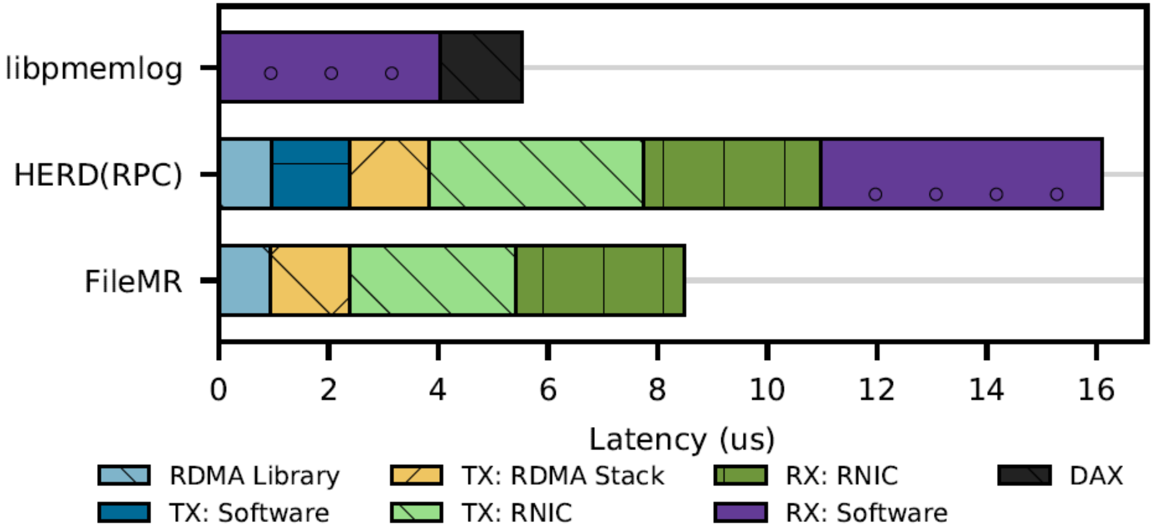


FileMR: Evaluation

- MR Registration time
  - The registration time of FileMR is much less (< 1%) than MR on file or shmem

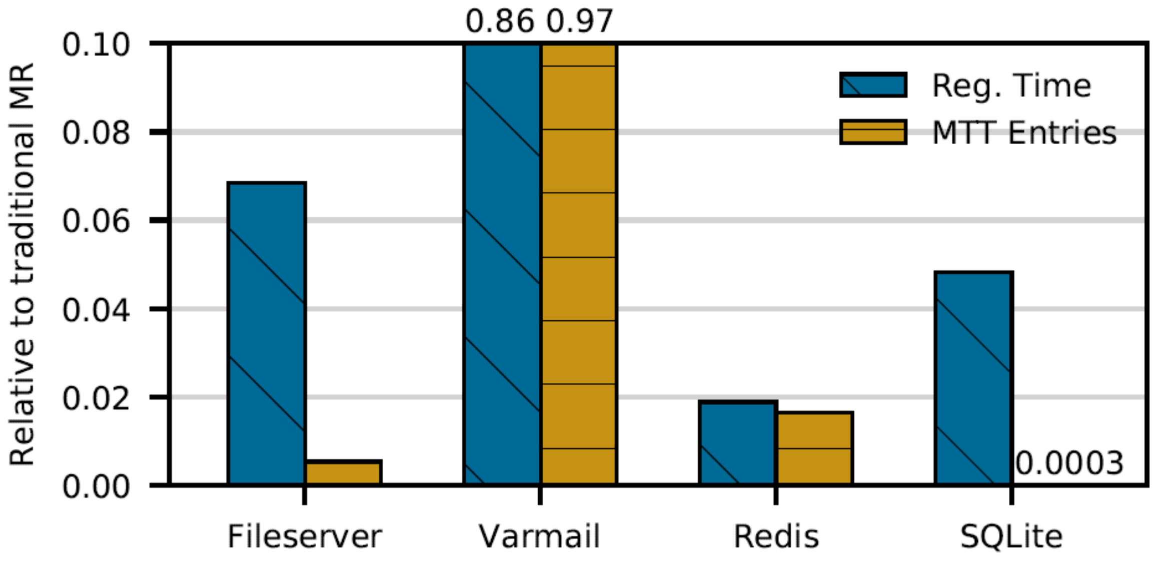


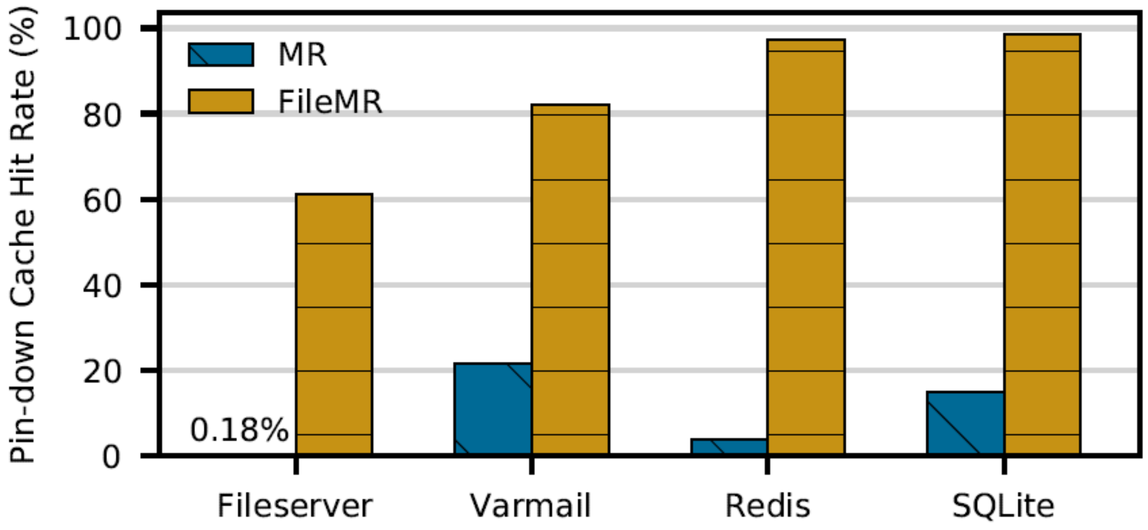
- Log appending latency breakdown
  - FileMR adds 53% overhead over `libpmemlog`
  - HERD-RPC adds 192% overhead



RangeMTT: Evaluation

- FileMR+RangeMTT vs. Registered MR+MTT
  - Registration time saving (1.8% ~ 86.2%)
  - # MTT entries saving (0.03% ~ 97%) are less significant on fragmented files.
  - FileMR has higher cache hit rate for all workloads. (Hot files stay in cache)



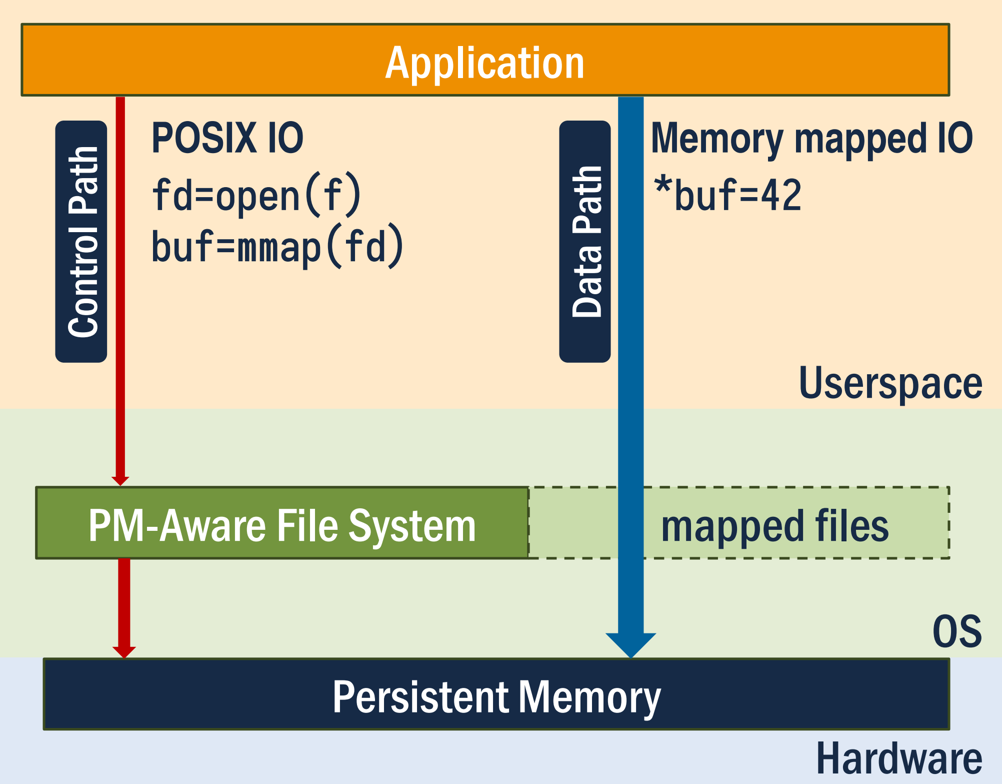


Conclusion

- Persistent memory provides byte-addressable memory accesses with persistency.
- RDMA networking enables fine-grained remote memory accesses.
- PM and RDMA should allow user to access remote PM directly, however:
  - PM and RDMA handle address translation in incompatible ways
  - Both PM and RDMA provide allocation, naming and permission checks
  - Existing user MR registration and address translation cause overhead
  - Existing user MR prevents PM from updating file layouts

- **FileMR**: using files as RDMA memory regions
- **RangeMTT**: leveraging file contiguity and translate file extents

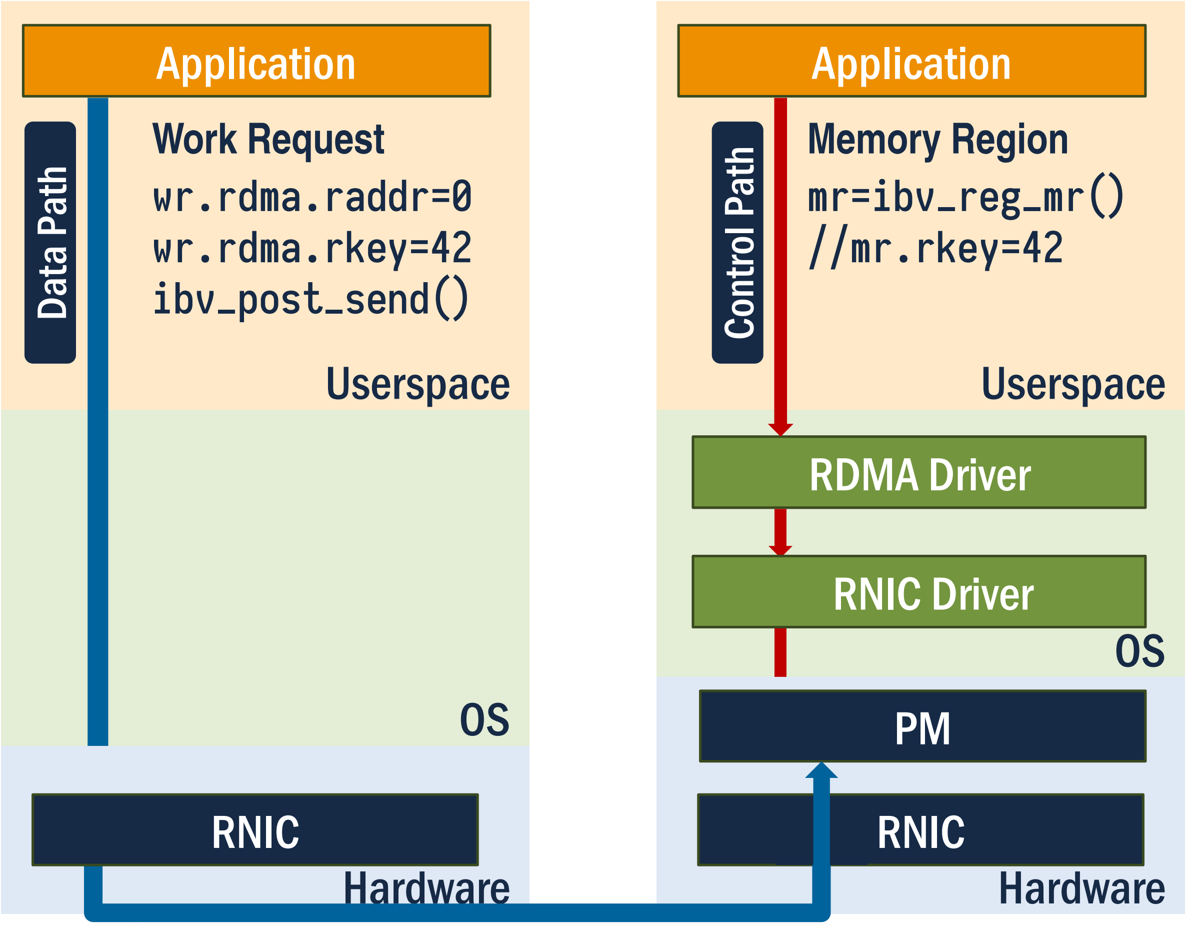
PM: memory management



- Allocation:
  - File system managed
  - Deferred: append
- Translation:
  - Contiguity: file extents
  - Dynamic: defragmentation (GC), transparent huge pages
- Protection:
  - File system managed (ACL)
- Naming:
  - Persistent hierarchical files
  - System-wide

NVSL

RDMA memory management



- Allocation:
  - Application managed (RDMA context)
- Translation:
  - Part of virtual address translation
  - Static: pinned pages
- Protection:
  - Protection domain (PD)
- Naming:
  - Implicit naming
  - PD-wide