Leveraging Non-Volatile Main Memory to store the state of Cloud applications reliably

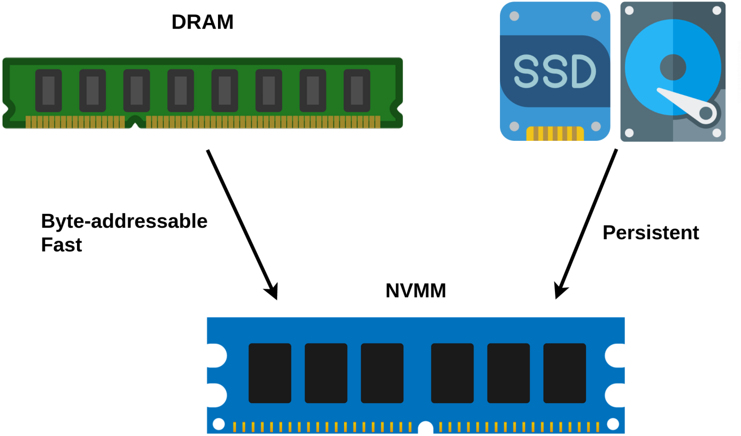
Thomas Ropars
Ana Khorguani Ivane Adam

Univ. Grenoble Alpes

2024

Non-Volatile Main Memory (NVMM)

NVRAM, NVDIMM



DRAM

Byte-addressable
Fast

Persistent

NVMM

- Intel-Micron 3D XPoint memory modules (2017)

A huge opportunity

Building highly efficient Fault-tolerant Cloud Applications

- Any interactive application
- In-memory KV store
- Our focus: Multi-threaded applications

A huge opportunity

Building highly efficient Fault-tolerant Cloud Applications

- Any interactive application
- In-memory KV store
- Our focus: Multi-threaded applications

Some challenges

- Performance not on par with DRAM (6-8X slower write throughput)
- Intermediate caches can impair data persistence and consistency

Work directions

Main research questions

- What API should be provided to programmers?
- What technique to efficiently save data in NVMM?
  - In a server including only NVMM
  - In a hybrid DRAM-NVMM server
  - In a remote NVMM server

Main results

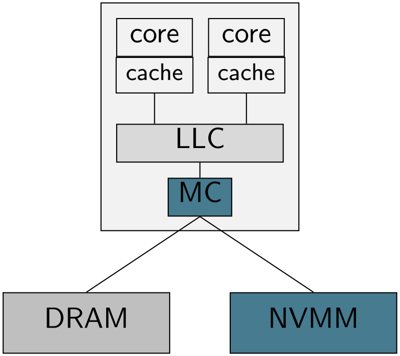
About APIs

- Providing a simple API to programmer provides huge advantages compared to *transparent* solutions
    - Concept of Restart Points

Techniques to save data

- NVMM-only: InCLL combined with Restart Points is the best technique (Khorguani et al., Eurosys 2022)
- Hybrid servers: No single best technique
- Remote NVMM: Redo-logging seems to be the best approach (WIP)

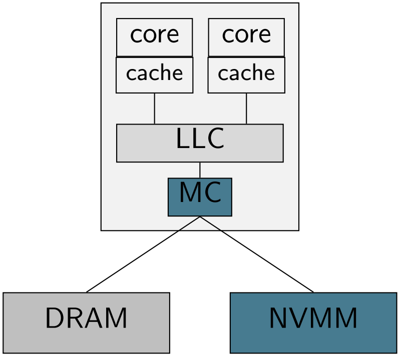
Using NVMM for fault tolerance



Consistent state and performance

- Data movements between the cache and the memory can be **controlled** by the application
    - Explicit flush of cache lines (Slow)

Using NVMM for fault tolerance



Consistent state and performance

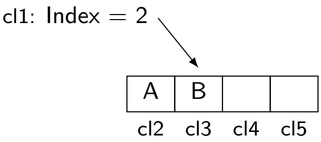
- Data movements between the cache and the memory can be **controlled** by the application
  - Explicit flush of cache lines (Slow)
- **On cache-line eviction, data might be written out-of-order to memory**

Cache-line eviction and consistent state

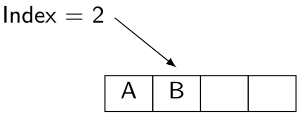
Produce in Producer-Consumer algo (FIFO Queue)

```
int index;
type buffer[SIZE];

void produce(type item){
  lock(&mutex)

  buffer[index] = item;
  index++;

  unlock(&mutex)
}
```

In cache:



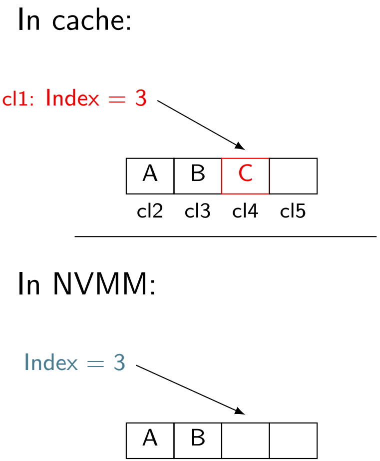
In NVMM:



Cache-line eviction and consistent state

Produce in Producer-Consumer algo (FIFO Queue)

```
int index;
type buffer[SIZE];

void produce(type item){
  lock(&mutex)

  buffer[index] = item;
  index++;

  unlock(&mutex)
}
```



In cache:

cl1: Index $= 3$

A	B	C	
cl2	cl3	cl4	cl5

In NVMM:

Index $= 3$

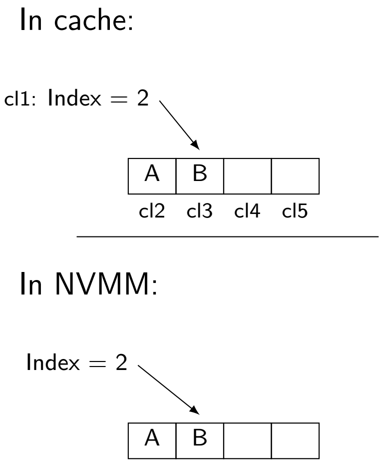
A	B		

The state in NVMM can become inconsistent

Cache invalidation guarantees a consistent state

Using clwb + MFENCE

```
int index;
type buffer[SIZE];

void produce(type item){
  lock(&mutex)

  buffer[index] = item;
  clwb(&buffer[index]);
  MFENCE
  index++;

  unlock(&mutex)
}
```



In cache:

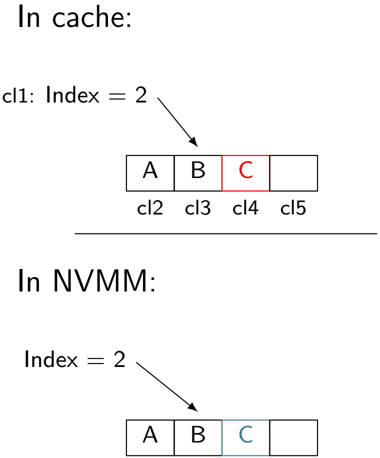
In NVMM:

Cache invalidation guarantees a consistent state

Using clwb + MFENCE

```
int index;
type buffer[SIZE];

void produce(type item){
  lock(&mutex)

  buffer[index] = item;
  clwb(&buffer[index]);
  MFENCE
  index++;

  unlock(&mutex)
}
```

Huge impact on performance


In cache:

cl1: Index = 2

A	B	C	
cl2	cl3	cl4	cl5

In NVMM:

Index = 2

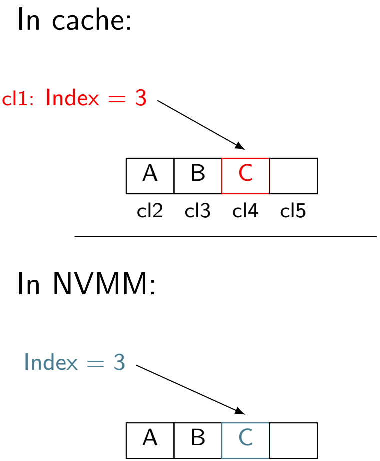
A	B	C	

Cache invalidation guarantees a consistent state

Using clwb + MFENCE

```
int index;
type buffer[SIZE];

void produce(type item){
  lock(&mutex)

  buffer[index] = item;
  clwb(&buffer[index]);
  MFENCE
  index++;

  unlock(&mutex)
}
```

Huge impact on performance



In cache:

cl1: Index = 3

A B C

cl2 cl3 cl4 cl5

In NVMM:

Index = 3

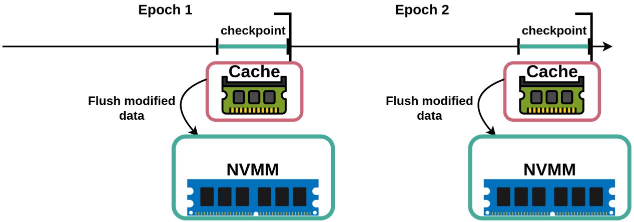
A B C

Solution for NVMM-only servers

ResPCT

Periodic synchronization with NVMM

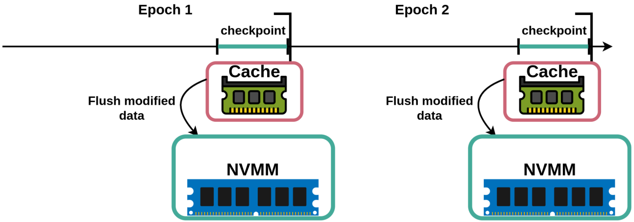
- High frequency checkpoints
- Flush modified data from cache to NVMM



ResPCT

Periodic synchronization with NVMM

- High frequency checkpoints
- Flush modified data from cache to NVMM



Programmers identify Restart Points

- Points in the execution where a checkpoint can be taken

```
void  produce(type item){
  lock(&mutex)
  buffer[index]  = item;
  index++;
  unlock(&mutex)

  RP()
}
```

In-Cache-Line Logging

Adapted from Cohen et al., ASPLOS 2019

An undo-log inside each cache line

- We fully avoid **Flush/Fence** instructions outside checkpoints
  - We take advantage of the x86 guarantees regarding writes the same cache line
- We allow some inconsistencies in NVMM
  - But we are always able to roll-back
- We also use InCLL to track modifications at no extra cost

In-Cache-Line Logging

Adapted from Cohen et al., ASPLOS 2019

An undo-log inside each cache line

- We fully avoid **Flush/Fence** instructions outside checkpoints
  - We take advantage of the x86 guarantees regarding writes the same cache line
- We allow some inconsistencies in NVMM
  - But we are always able to roll-back
- We also use InCLL to track modifications at no extra cost

Experimental setup

Hardware and software setup:

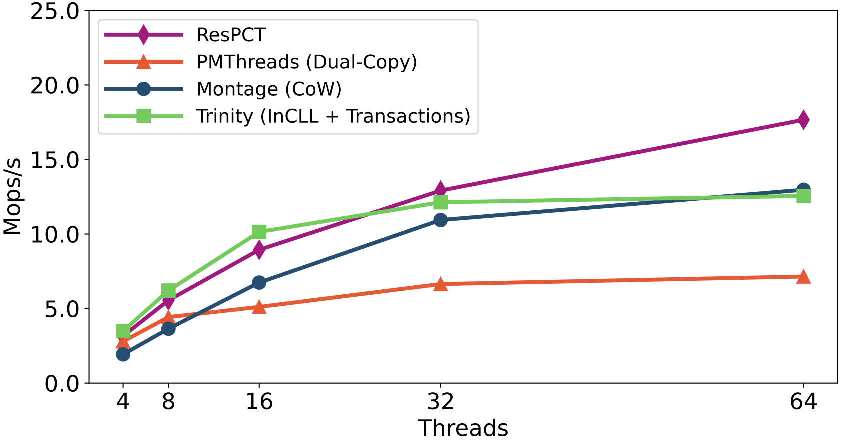
- A single server with two Intel Xeon Gold 5218 CPUs (64 logical cores)
- 384 GiB of DRAM and 1.5 TiB of Intel's Optane PMem
- Prototype of ResPCT in C
- Checkpoint period - 64 msec

Evaluated workloads:

- Highly efficient concurrent HashMap (2M items)
- Memcached - a popular in-memory key-value store

Results for the HashMap

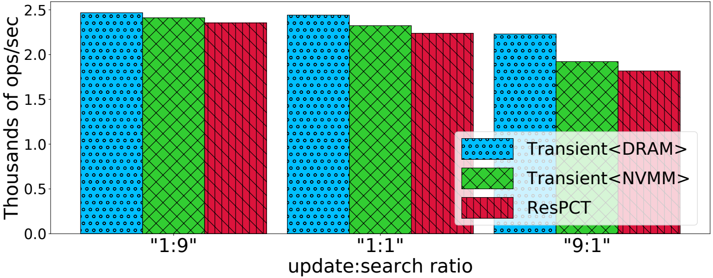
90% of updates



- ResPCT speed-up over best competitor: 1.36X
- ResPCT slowdown compared to non-modified hashmap: 8.3X

Performance with Memcached

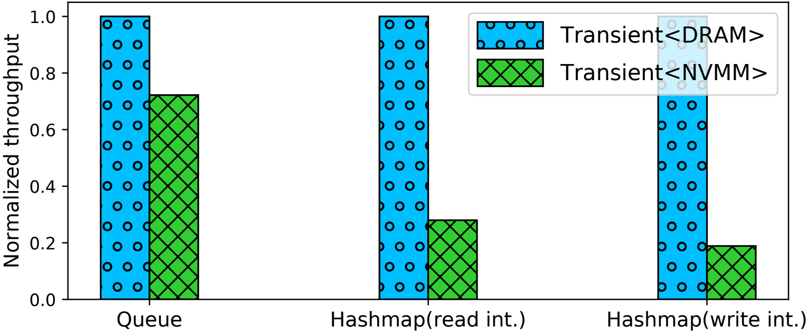
1M operations



- Overhead for the read-intensive workload: 5%
- Overhead for the write-intensive workload: 18.5%

The case of hybrid servers

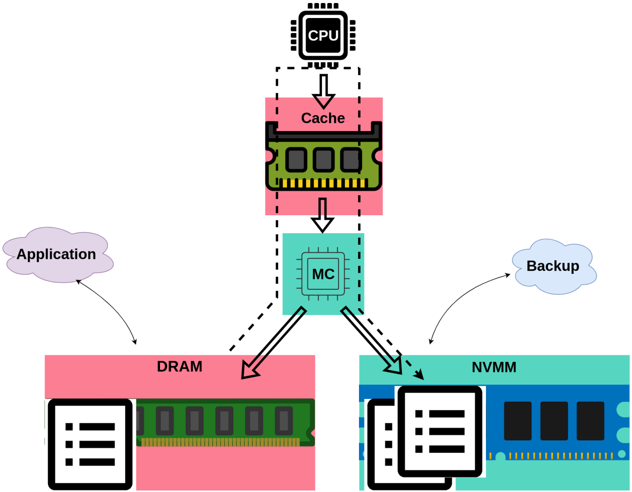
The limits of the NVMM-only approach



Performance is limited by the speed of NVMM

The hybrid approach

- The application interacts with DRAM
- NVMM stores backups
- Transfer from DRAM to NVMM during checkpoints
    - Transfer of the modified parts of the memory



New questions

- What technique to use for consistent data transfer?
  - No issue with cache-line invalidation but **the server might still crash in the middle of the transfer**
  - We evaluated the main approaches from SOA:
    - InCLL (Undo log)
    - Redo log [Aksun, EPFL 2021]
    - Dual copy [WU et al., PLDI 2020]

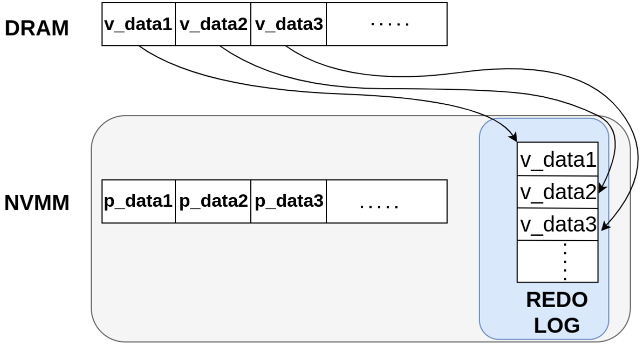
New questions

- What technique to use for consistent data transfer?
    - No issue with cache-line invalidation but **the server might still crash in the middle of the transfer**
    - We evaluated the main approaches from SOA:
        - InCLL (Undo log)
        - Redo log [Aksun, EPFL 2021]
        - Dual copy [WU et al., PLDI 2020]

- What granularity to use for tracking modifications?
- What granularity to use for flushing modifications?

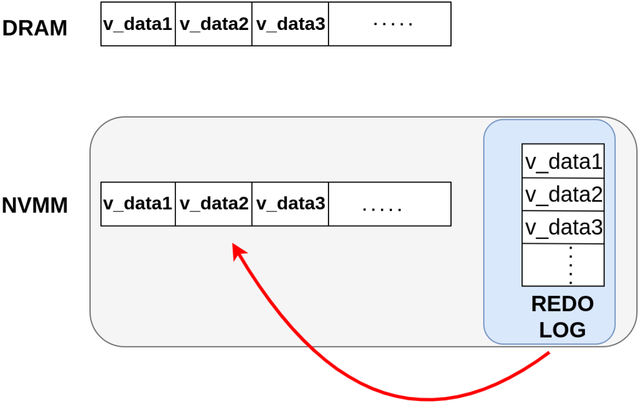
The Redo-Logging approach

Checkpointing: Write a redo log of modifications to NVMM



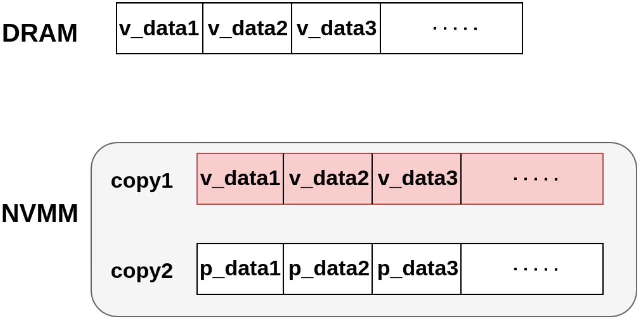
The Redo-Logging approach

The state update is done in the background



The Dual-Copy approach

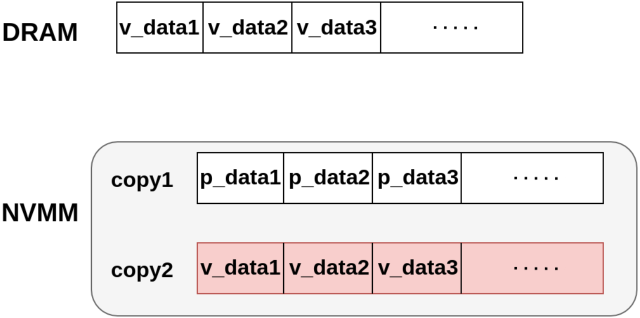
One copy is updated in a given checkpoint



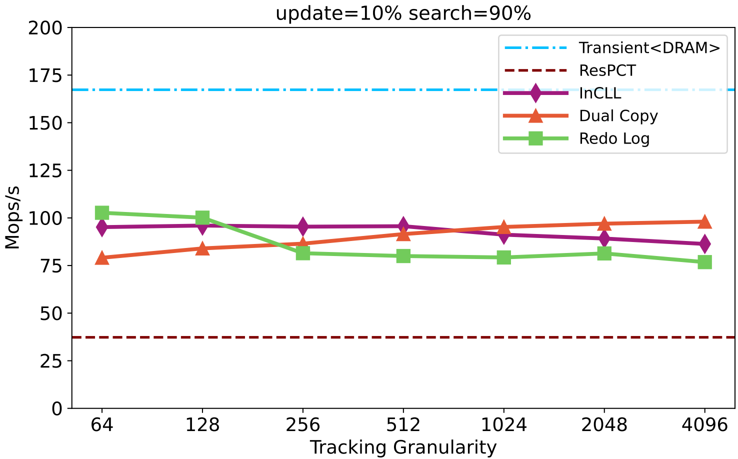
DRAM		v_data1	v_data2	v_data3	· · · · ·
NVMM	copy1	v_data1	v_data2	v_data3	· · · · ·
	copy2	p_data1	p_data2	p_data3	· · · · ·

The Dual-Copy approach

The other copy is updated in the next checkpoint (The previous copy becomes the backup)

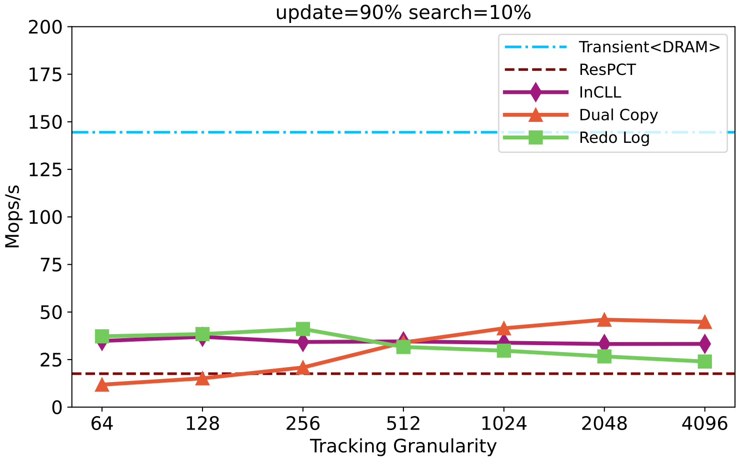


Results for the HashMap (Read-intensive workload)



- Best results with Redo Logging
- Slowdown to Transient: 1.6X – Speedup to ResPCT: 2.7X

Results for the HashMap (Write-intensive workload)



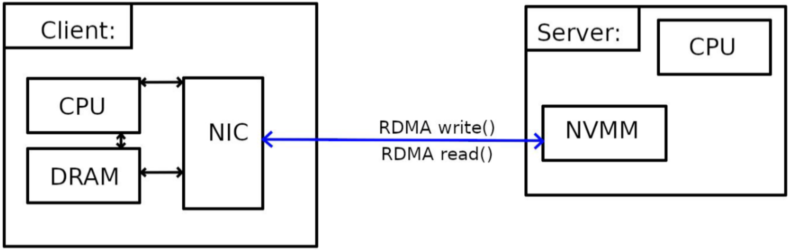
- Best results with Dual Copy
- About the tracking granularity:
  - Small: More overhead for tracking, less for flushing
  - Large: Less overhead for tracking, more for flushing

Remote NVMM

Saving data in Remote NVMM

RDMA writes to NVMM can be made persistent

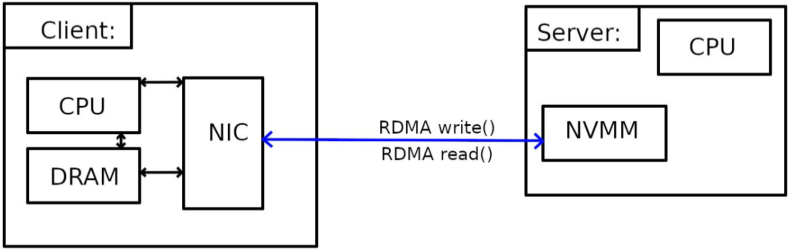
- By de-activating DDIO
- Issuing a flush after the write operation



Saving data in Remote NVMM

RDMA writes to NVMM can be made persistent

- By de-activating DDIO
- Issuing a flush after the write operation

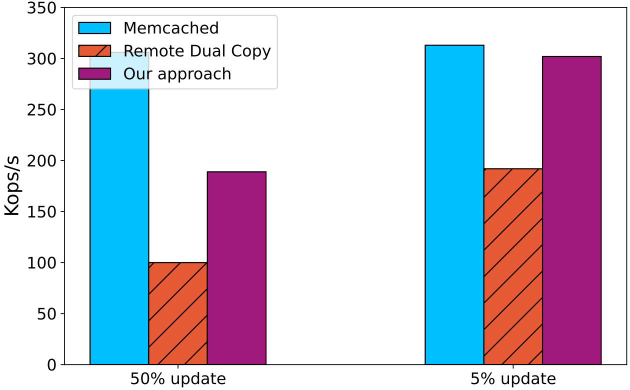


Our approach

- Same checkpointing approach as before
- Algorithm:
    - Writing a Redo-Log to Remote NVMM using RDMA
    - Updating the persistent state in the background

Preliminary results with MemCached (1M Keys)

Network used: Omni-Path 100G



- Important for performance: Writing large memory blocks
  - Otherwise performance is limited by the network latency

Conclusion

Main results

A new approach for saving application state to NVMM

- Periodic checkpoints
- Restart Points specified by the programmer

Performance

- Better performance than SOA:
  - NVMM-only: InCLL
  - Hybrid servers: Dual Copy or Redo Logging
- Best technique depends on the considered hardware architecture
  - Redo Logging is the most promising for Remote NVMM

Future Directions

Consider other technologies (PEPR Cloud)

- NVMe
  - Ability to do remote writes directly to NVMe devices?
  - WiP: Go through an intermediate copy in remote DRAM

- CXL memory expanders
  - Support for flush operations included
  - Expanders start appearing

Leveraging Non-Volatile Main Memory to store the state of Cloud applications reliably

Thomas Ropars
Ana Khorguani Ivane Adam

Univ. Grenoble Alpes

Université
Grenoble Alpes