Improving Virtualized I/O Performance by Expanding the Polled I/O Path of Linux

Dongjoo Seo[1], Yongsoo Joo[2], Nikil Dutt[1]

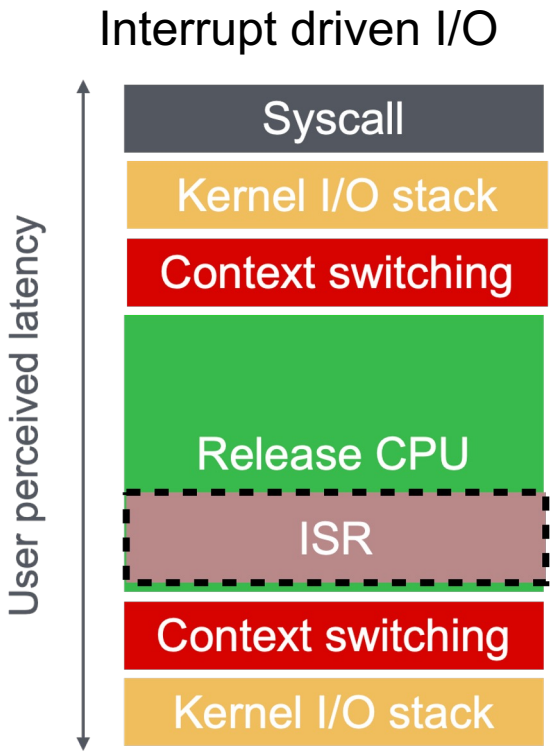
[1]UC Irvine, CA

[2]Kookmin Univ., Seoul, Korea

UCIRVINE

HotStorage'24

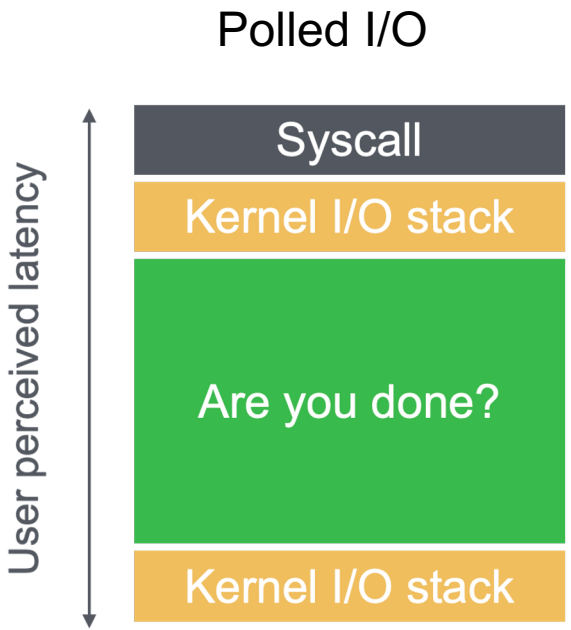
Interrupt: I/O Completion Method for Storage



- Interrupt adds delay to the raw device latency
- **Context switching** and **ISR** are the primary sources of the additional delay
  - Typically 2 µs or more
- The overhead varies by the type of storage

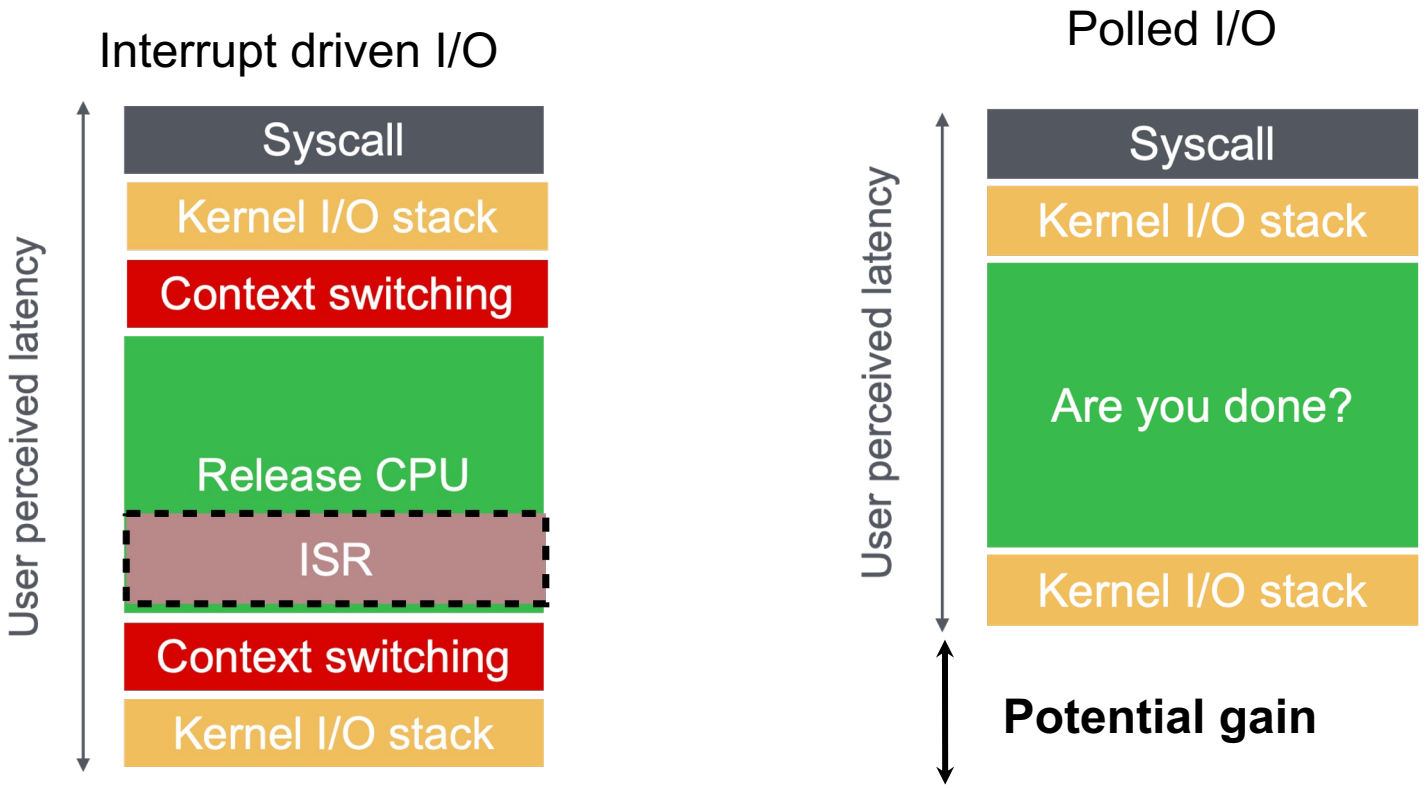
4 KB random read latency		Overhead (typ.)
HDD	2~20ms	0.001~0.0001%
TLC SSD	40~50 us	2~3%
ULL SSD	8~20 us	10~25%

I/O Polling: An Alternative to Interrupts



- **Continuously checks** the I/O completion queue until the I/O operation is complete
- No context switching, ISR, or CPU sleep and wakeup is required

Interrupt-Driven I/O vs. Polled I/O



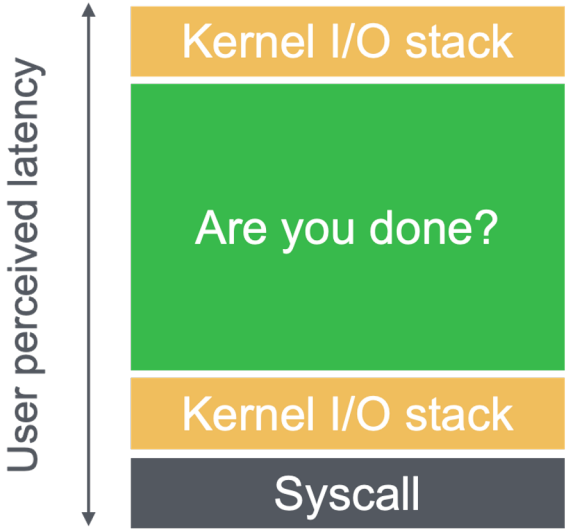
Challenges Limiting the Adoption of I/O Polling

- #1: Low popularity of ULL SSDs
  - High cost per bit
  - Manufacturers are winding down their ULL SSD businesses

- #2: Limited support of Linux
  - Currently, the Linux polled I/O path is only accessible via io_uring
  - Direct sync I/O has been removed since kernel 5.19
  - No support for buffered I/Os and memory mapped I/Os

Challenge #1: Finding Alternatives to ULL SSDs

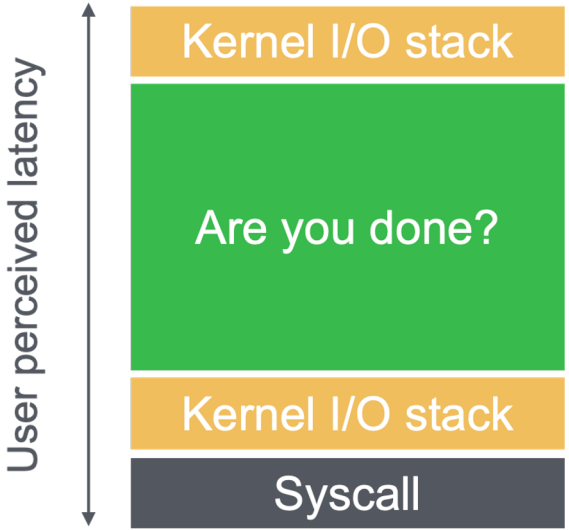
- Consider DRAM as an alternative to ULL SSDs:
  - DRAM is much faster than ULL SSDs
  - Potential gain from using I/O polling should be greater as well



4 KB random read latency		Potential gain
HDD	2~20ms	0.001~0.0001%
TLC SSD	40~50 us	2~3%
ULL SSD	8~20 us	10~25%
What about DRAM?	**?**	**?**

Challenge #1: Finding Alternatives to ULL SSDs

- Consider DRAM as an alternative to ULL SSDs:
  - DRAM is much faster than ULL SSDs
  - Potential gain from using I/O polling should be greater as well



4 KB random read latency		Potential gain
HDD	2~20ms	0.001~0.0001%
TLC SSD	40~50 us	2~3%
ULL SSD	8~20 us	10~25%
What about DRAM?	**Less than < 200 ns**	**> 10X**

Can We Use DRAM as a Block Device?

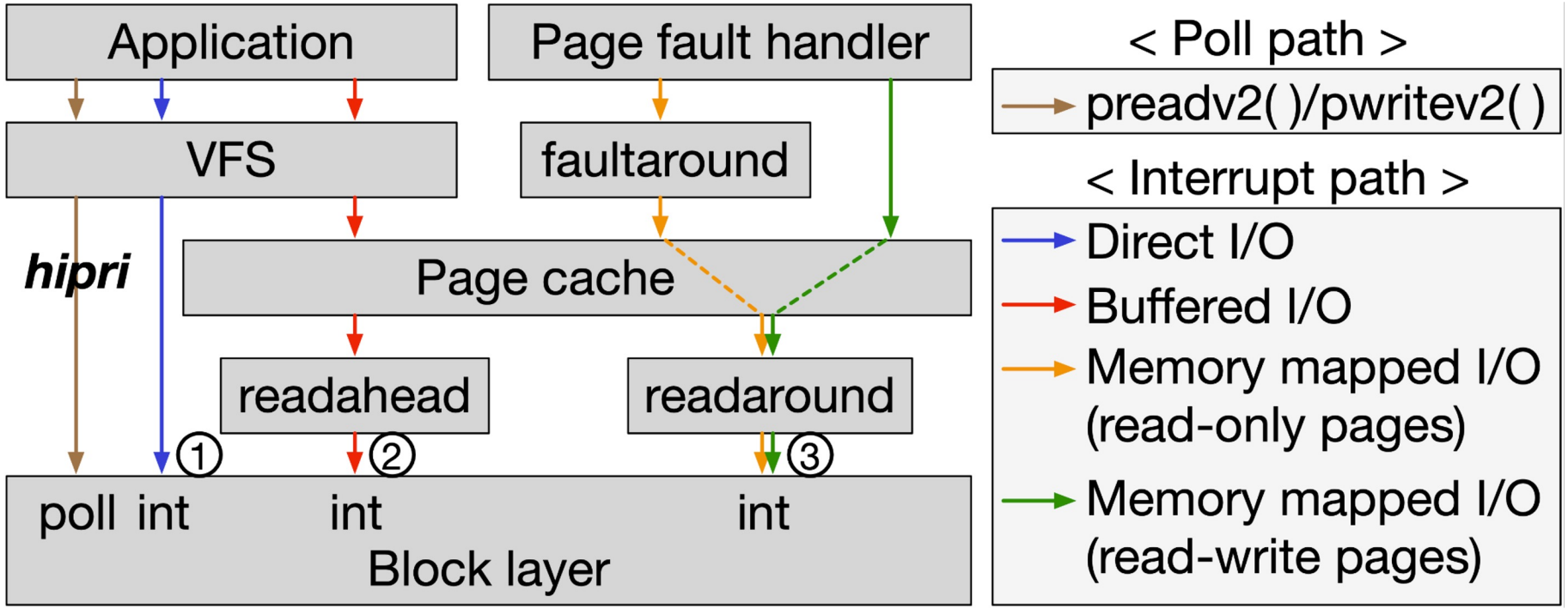
- Special cases when DRAM is accessed using I/O requests:
  - Battery backed DRAM SSDs
    - Share the same limitation as ULL SSDs
  - DRAM host cache in virtualized environments
    - Guest applications issue I/O requests to access the host cache
    - Guest OSes rely on interrupts even when accessing the host cache

Can We Use DRAM as a Block Device?

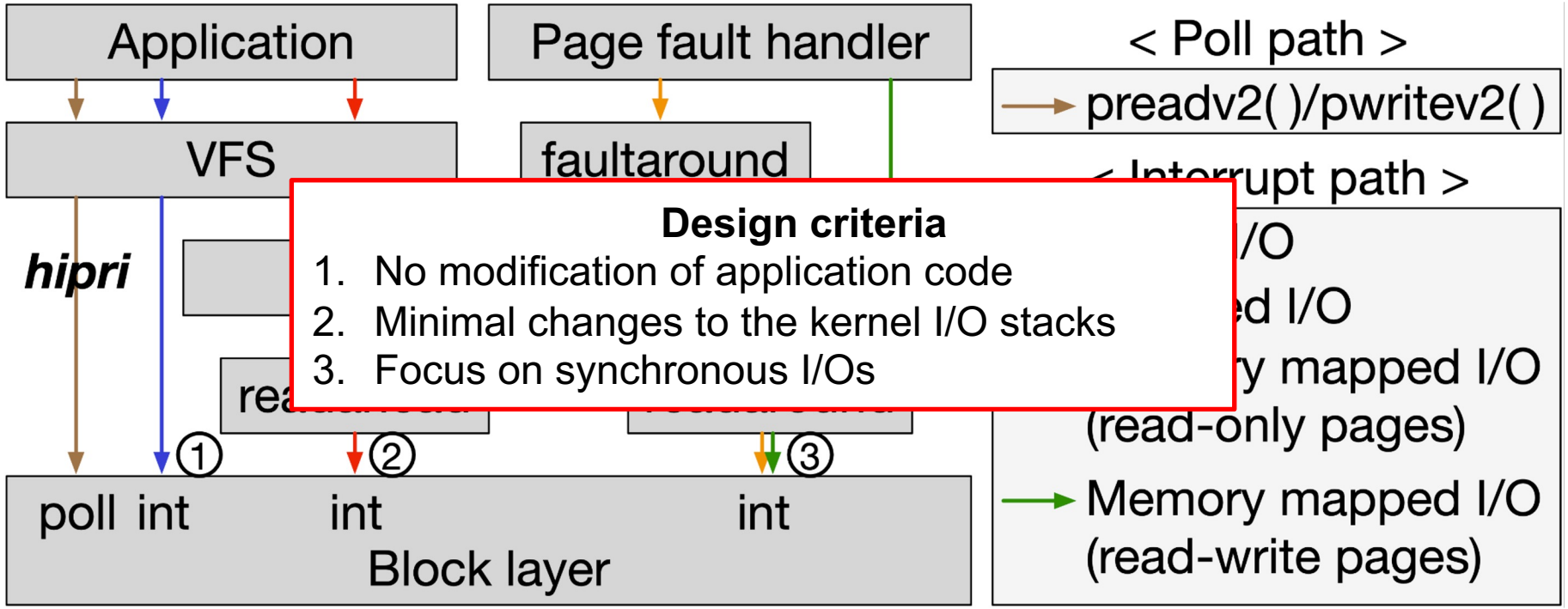
- Special cases when DRAM is accessed using I/O requests:
  - Battery backed DRAM SSDs
    - Share the same limitation as ULL SSDs
  - DRAM host cache in virtualized environments

Proposal to Challenge #1: Finding Alternatives to ULL SSDs: Apply I/O polling for DRAM host cache

Challenge #2: Enhancing Kernel Support for I/O Polling

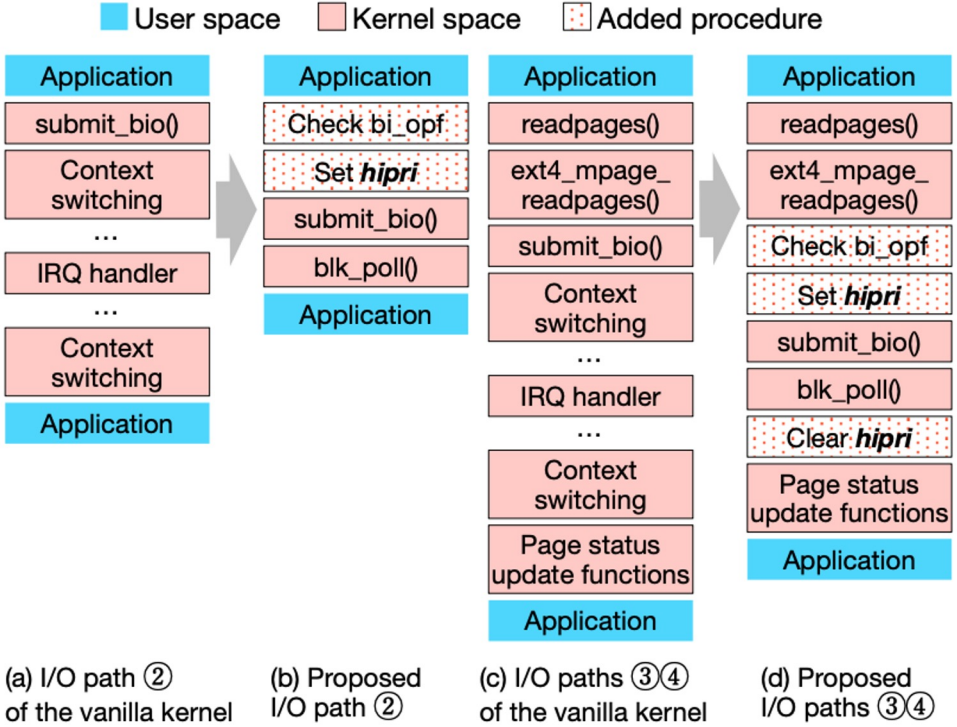


Challenge #2: Enhancing Kernel Support for I/O Polling



Proposal to Challenge #2: Flag Injection

- I/O polling codes in Linux are activated with the hipri flag
- ***hipri*** flag injection enables I/O polling code reuse with minimal Linux block layer modification
- Newly added support for I/O polling:
  - Sync I/Os using read() & write()
  - Memory-mapped I/Os

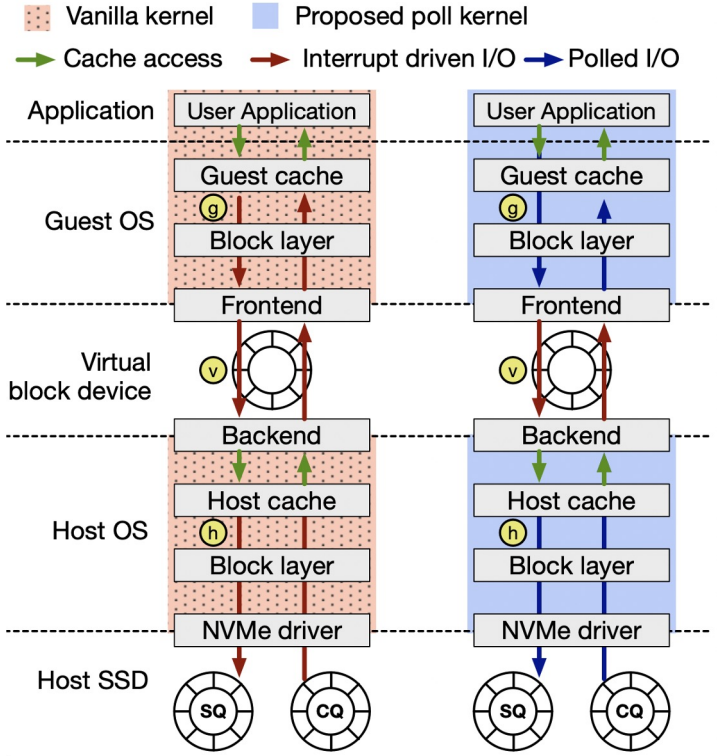


Diff Summary

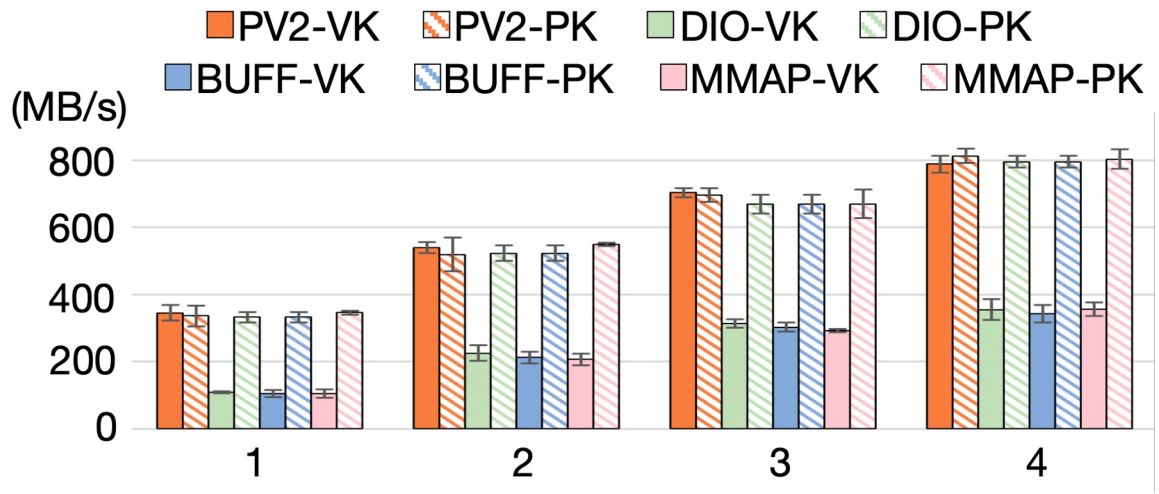
```
 drivers/nvme/host/core.c        |  3 ++-
 fs/ext4/file.c                  |  2 +-
 fs/ext4/readpage.c              | 57 ++++++++++++++++++++++++++++++++++++++++++++++++---------
 fs/iomap/direct-io.c            |  3 +--
 block/bio.c                                |  2 ++
 block/blk-core.c                |  3 +++
 block/blk-merge.c               |  2 +-
 block/blk-mq.c                             |  5 ++++-
 block/fops.c                               |  7 ++++---
 include/blk_types.h             |  4 ++++
 include/blkdev.h                |  3 +++
 mm/filemap.c                               |  2 +-
12 files changed, 74 insertion(+), 19 deletion(-)
```

Virtualized System Configurations

- Existing configuration:
  - Interrupt-driven I/O stack for both guest and host OSes
- Proposed polled I/O stack:
  - Replaces most of the interrupt-driven I/O path with polling, except for the VBD



Improving Host Cache Throughput

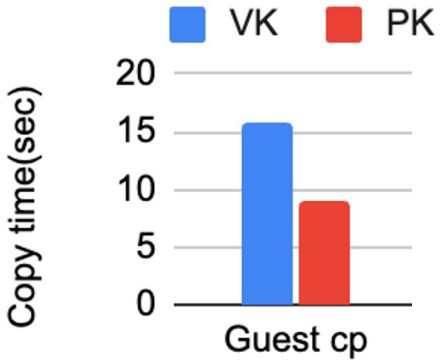


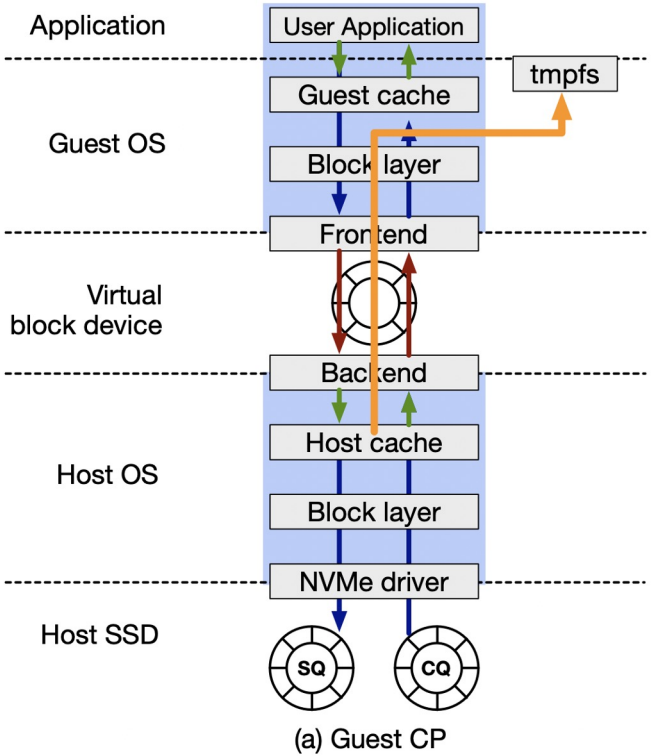
4kb random read @1~4 threads

I/O engine	Vanilla kernel (VK)		Poll kernel (PK)
	Interrupt	Polling	Polling
pvsync2 w/ `hipri`	N/A	PV2-VK	PV2-PK
sync w/ direct I/O	DIO-VK	N/A	DIO-PK
sync w/ buffered I/O	BUFF-VK	N/A	BUFF-PK
mmap	MMAP-VK	N/A	MMAP-PK

Copying Multiple Small-Sized Files

- File size: 4 KB
- # of files: 262,144 (total 1 GB)
- Guest CP: Host cache to guest tmpfs

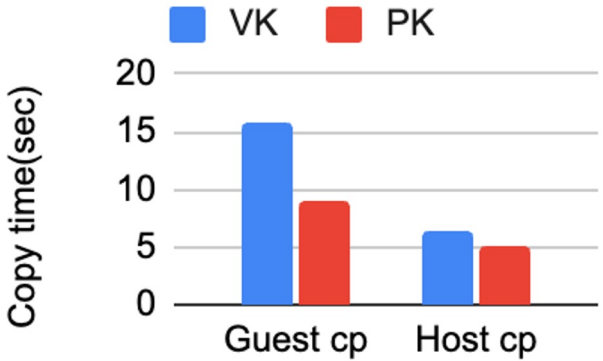


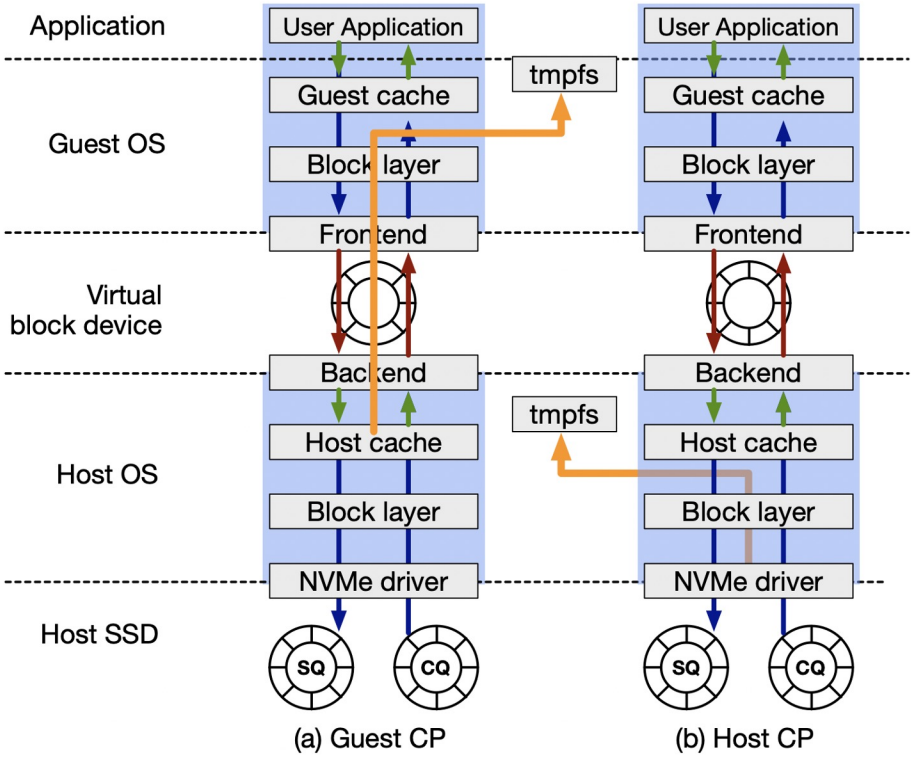


(a) Guest CP

Copying Multiple Small-Sized Files

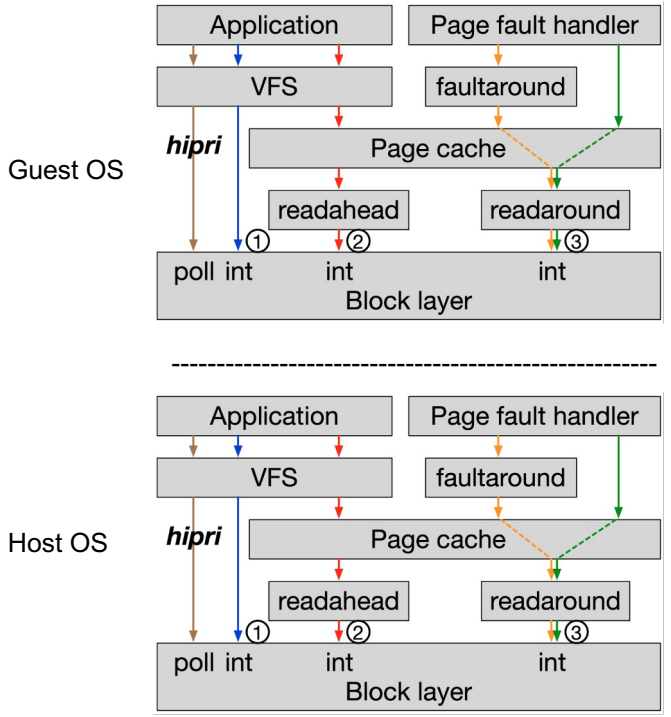
- File size: 4 KB
- # of files: 262,144 (total 1 GB)
- Guest CP: Host cache to guest tmpfs
- Host (ULL) SSD: Intel Optane 900P
- Host CP: Host SSD to host tmpfs





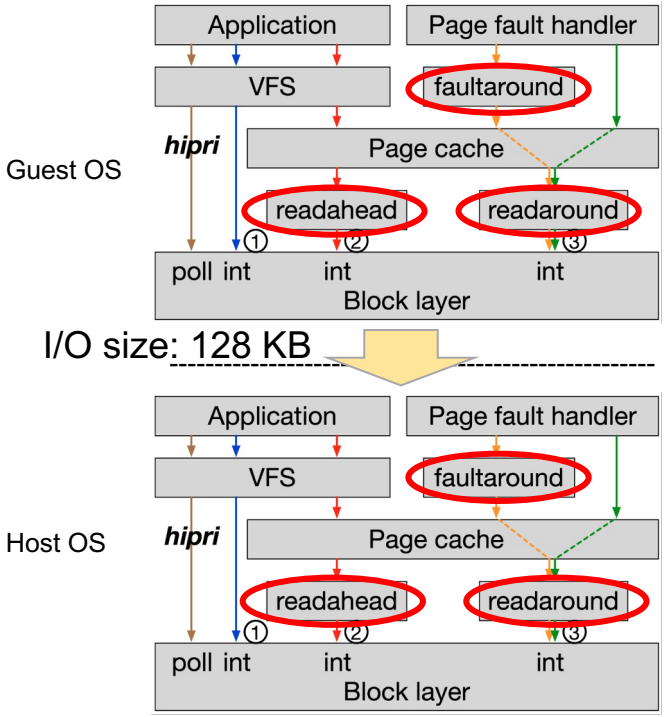
Guest App Launch from Host SSD

- Assume cold start scenario
  - App code and data fetched from the host SSD
- App launch well optimized thanks to kernel features:
  - Faultaround, readahead and readaround

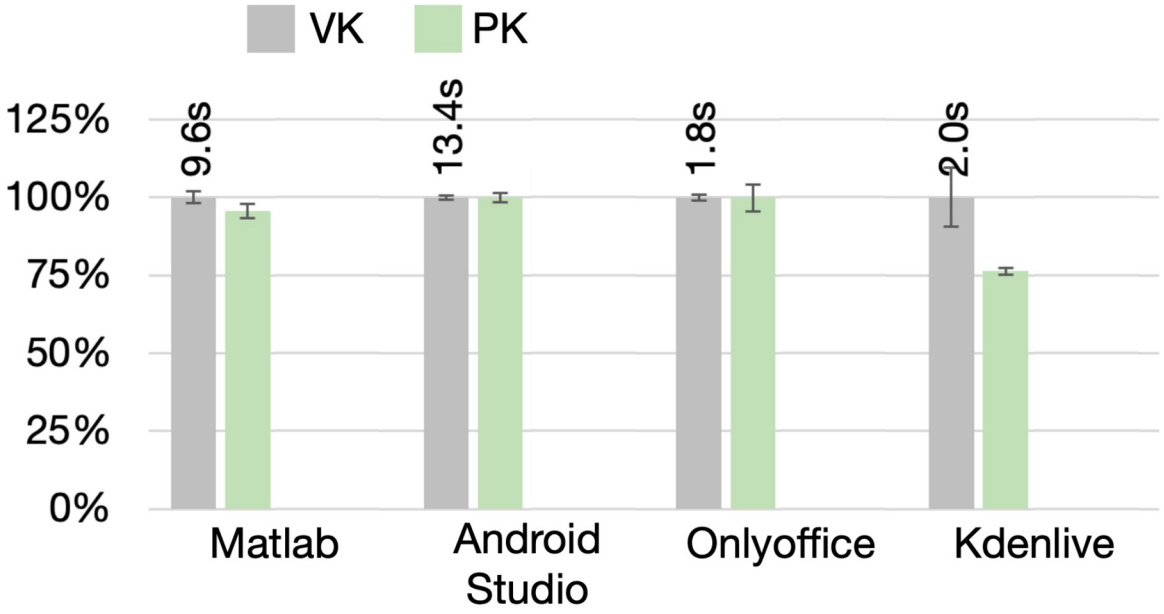


Guest App Launch from Host SSD

- Kernel features enabled (default)
- Utilizing faultaround, readahead and readaround
  - I/Os from the guest OS: **128 KB (typ.)**
  - App launch primarily **CPU-bound**

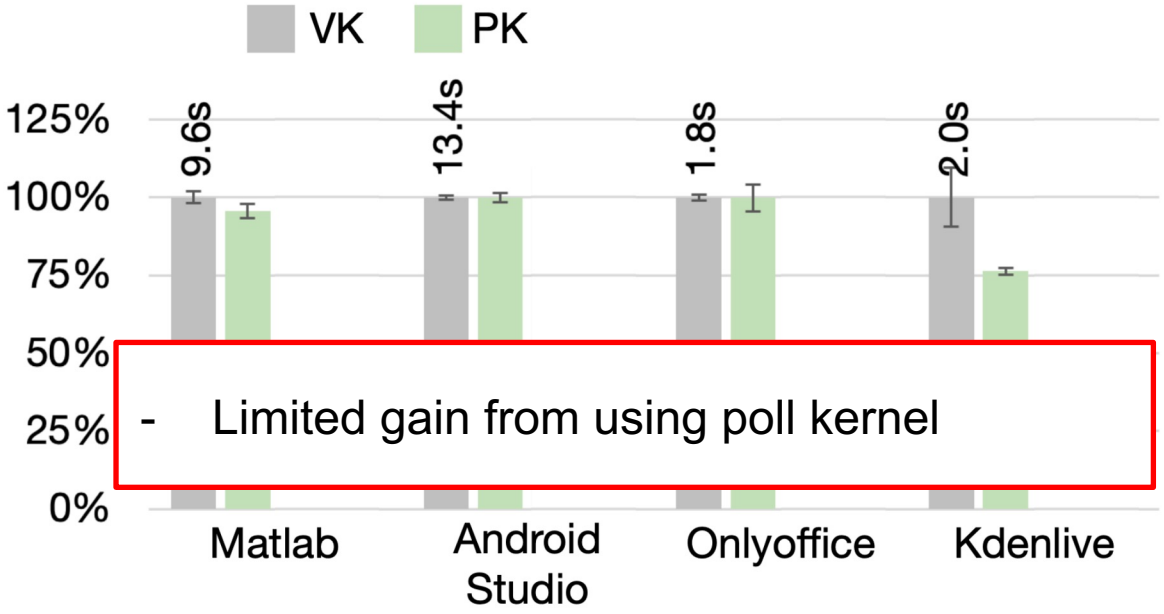


Guest App Launch from Host SSD



Normalized application launch time

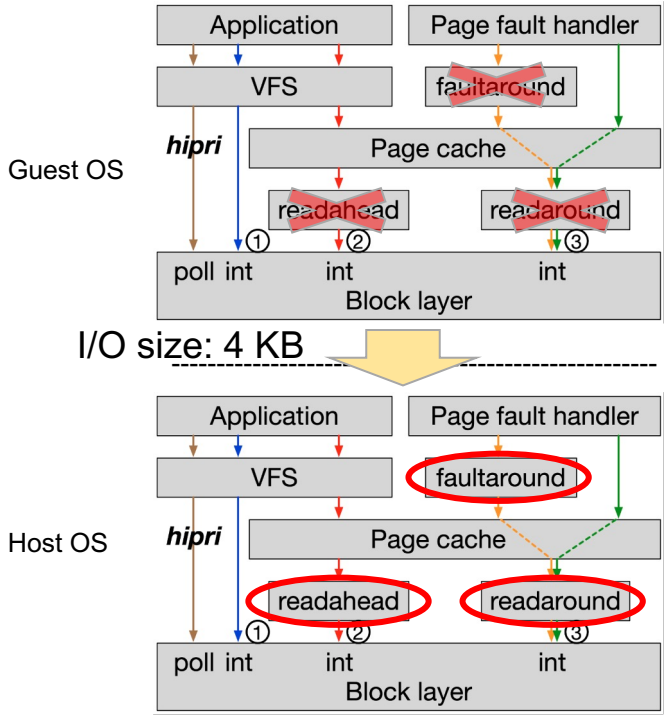
Guest App Launch from Host SSD



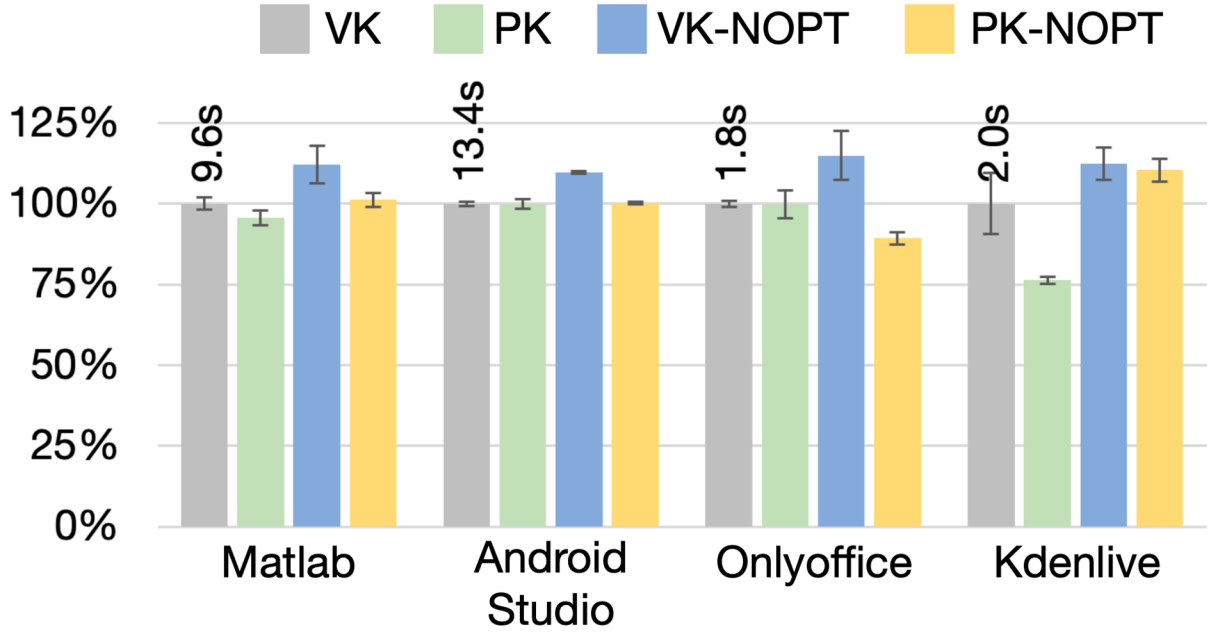
Normalized application launch time

Guest App Launch from Host SSD

- Kernel features disabled in **the guest OS** (NOPT)
  - I/Os from the guest OS: **4 KB** (typ.)
  - App launch becoming more **I/O bound**

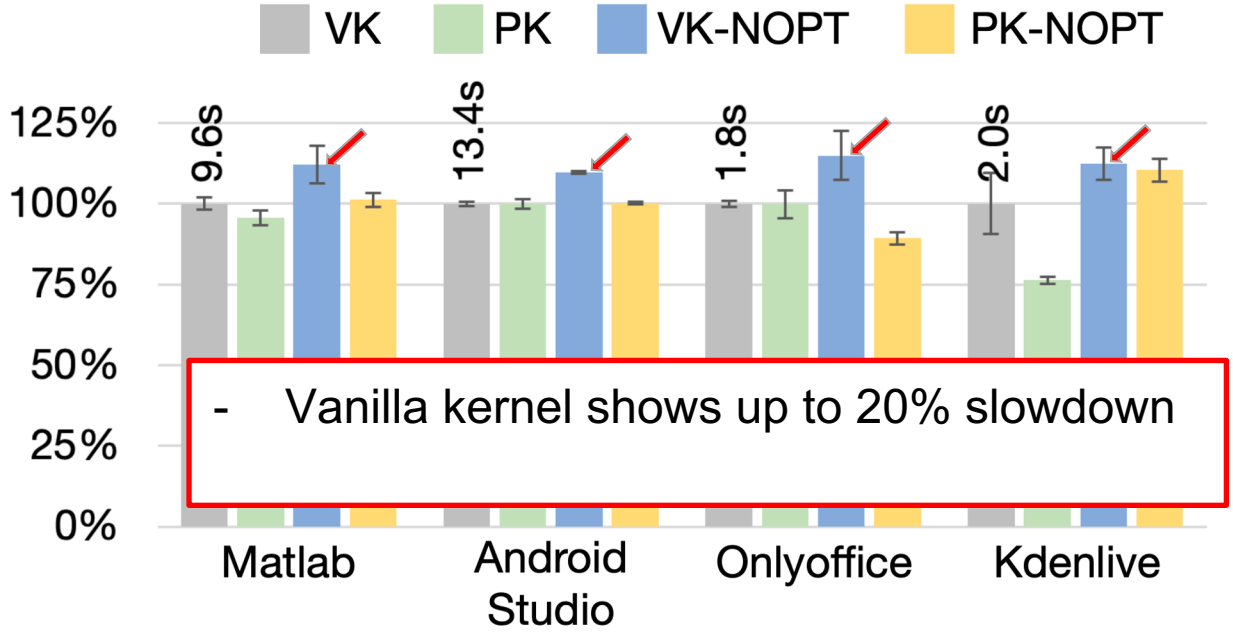


Guest App Launch from Host SSD



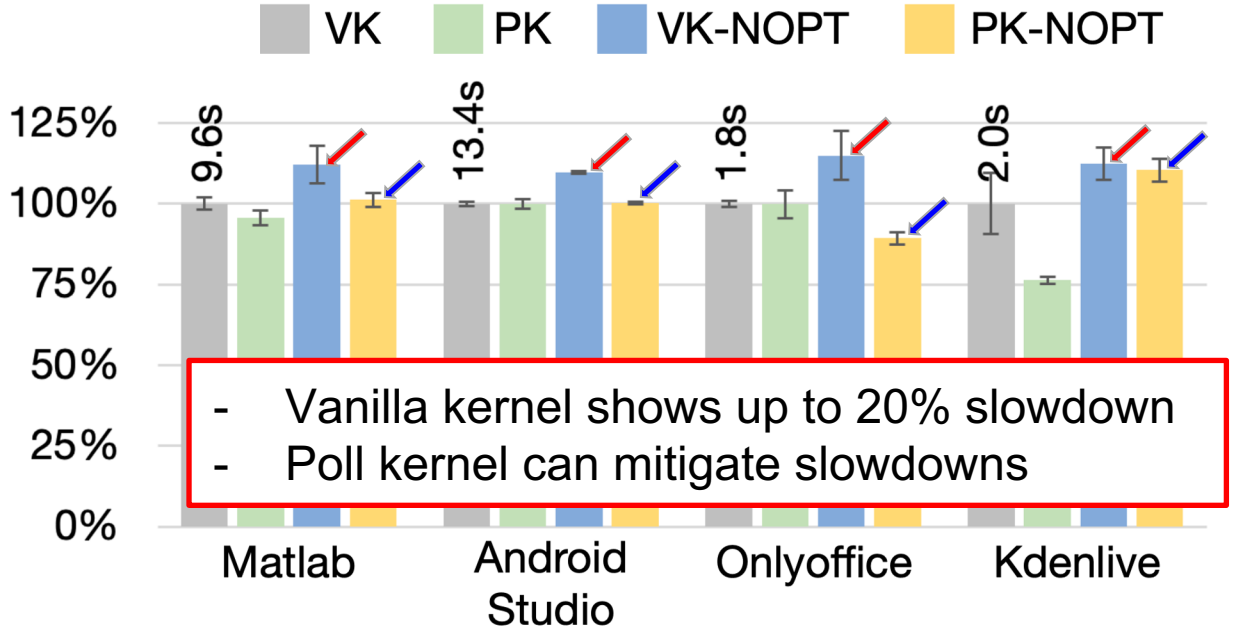
Normalized application launch time

Guest App Launch from Host SSD

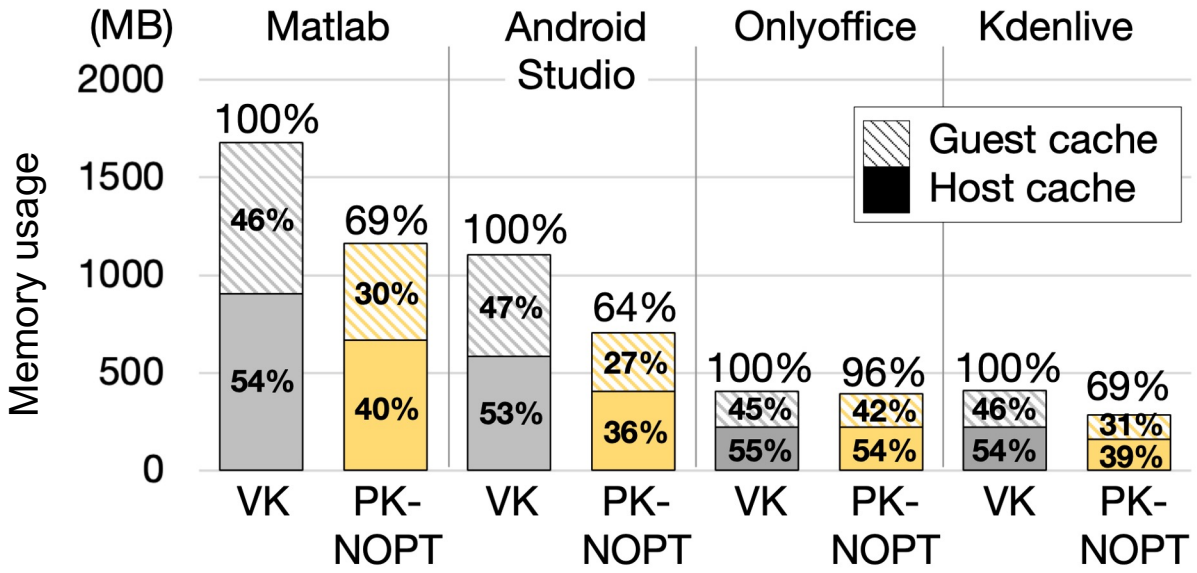


Normalized application launch time

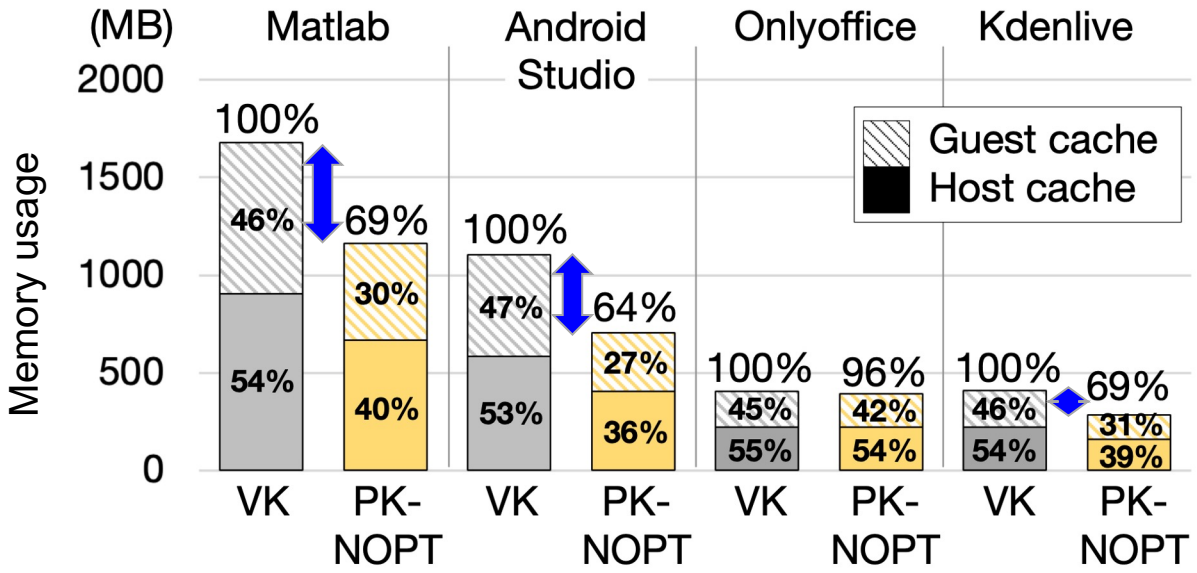
Guest App Launch from Host SSD



Memory Usage Reduction



Memory Usage Reduction



Memory usage reduction: 31% 36% 4% 31%

Conclusion

- Polled I/O path expansion
  - Benefits user applications without source modification

- Demonstrating polling benefits
  - Not limited to ULL SSDs
  - Improving access speed of host page cache by up to 3X in virtualized systems

- Use cases
  - Improving small file copy performance
  - Memory usage reduction for guest application startup

Future Work

- Further improving host page cache throughput
- Plan to identify stronger use cases
- Assessing the effects of CPU contention on polled I/O performance

Future Work

- Further improving host page cache throughput
- Plan to identify stronger use cases
- Assessing the effects of CPU contention on polled I/O performance

Thank you!