Extension framework for File systems in User space

Ashish Bijlani and Umakishore Ramachandran

Georgia Institute of Technology

USENIX ATC 2019

Background

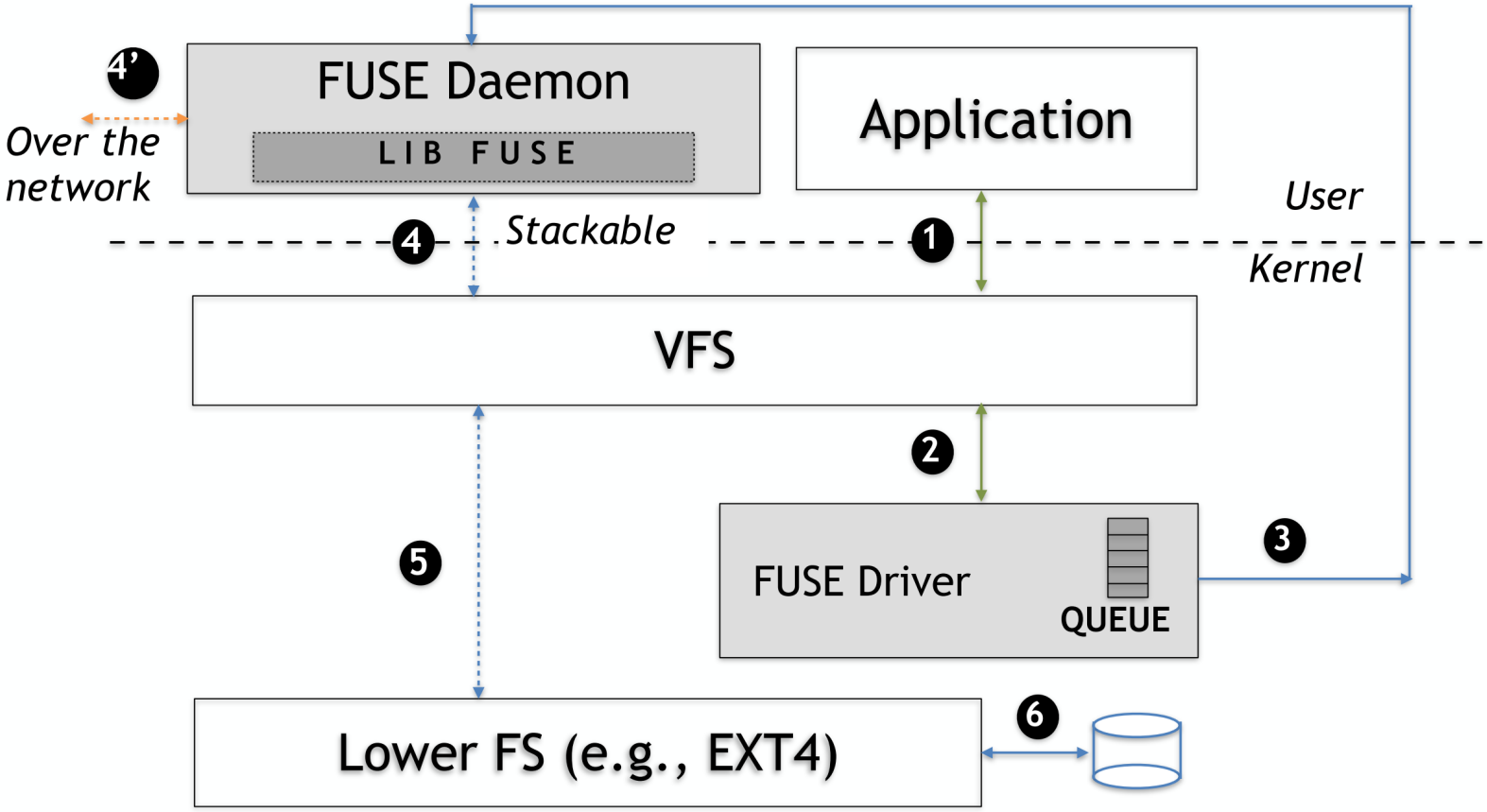
- **Kernel file system**
  - All file system handlers implemented in kernel space
  - Ext4, NTFS, APFS, FAT32…
- **Strength:**
  - Native performance
- **Weakness:**
  - Poor security,reliability
  - Not easy to develop, debug,maintain

- **User file system (FUSE)**
  - All file system handlers implemented in user space
  - GlusterFS, EncFS, S3FS
- **Strength:**
  - Improved security, reliability
  - Easy to develop, debug, maintain
- **Weakness:**
  - Poor performance

Background

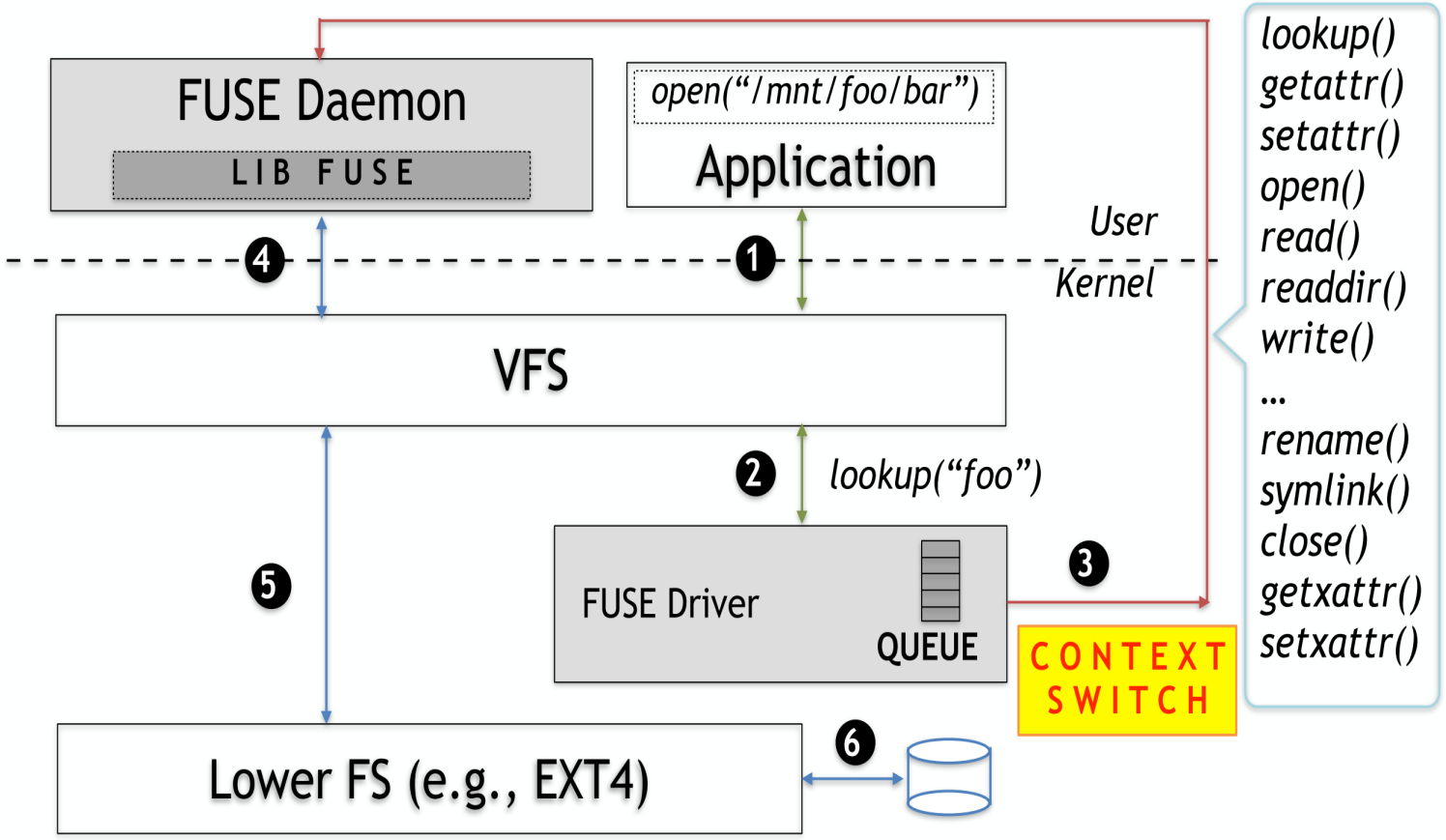
➢ FUSE Architecture

- **Virtual file system** (**VFS**) : allow client applications to access different types of concrete file systems in a uniform way



Motivation

- Challenge:
  - Frequent user-kernel round-trip communication and data copying
  - Overhead is higher with faster storage media (83% overhead for metadata-heavy workloads on SSD)



Motivation

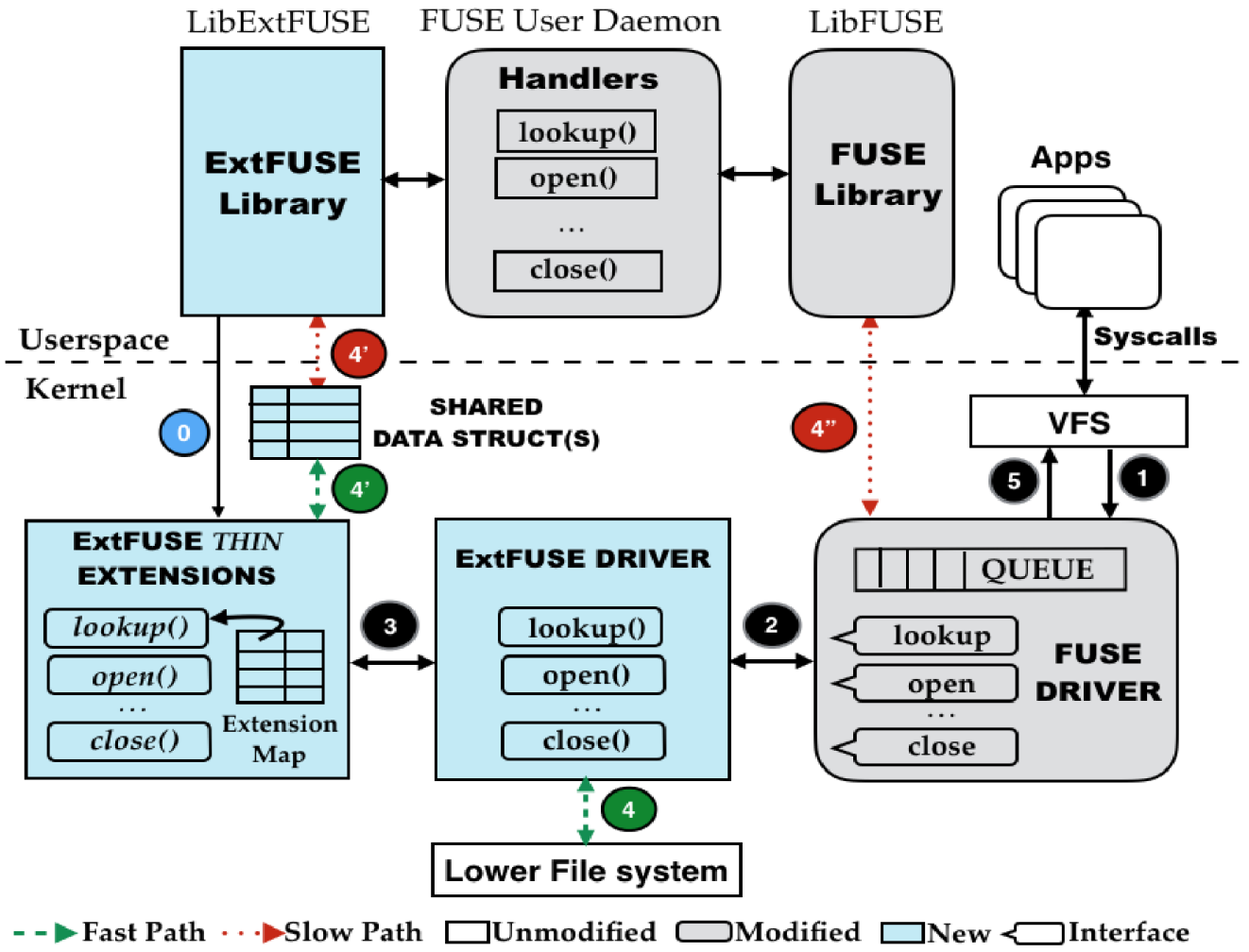
- Current Solutions
  - Zero-copy data transfer (splicing)
    - Only used for over 4K requests
    - Small writes/reads will incur data copying overheads
  - Utilizing system-wide shared VFS caches
    - Improves I/O throughput by batching requests, but no help to random reads while introduce higher write latency
- Biggest Problem: How to balancing safety and performance?

Contributions

- Identify optimization opportunities in user file system frameworks for monolithic OSes and propose extensible user file systems
- Introduce ExtFUSE, an extension framework for user file systems that offers the performance of kernel file systems, while retaining the safety properties of user file systems
- Adopt ExtFUSE to FUSE, and evaluating it on Linux
- Show benefits and limitations of the ExtFUSE with **Android sdcard daemon** and **LoggedFS**

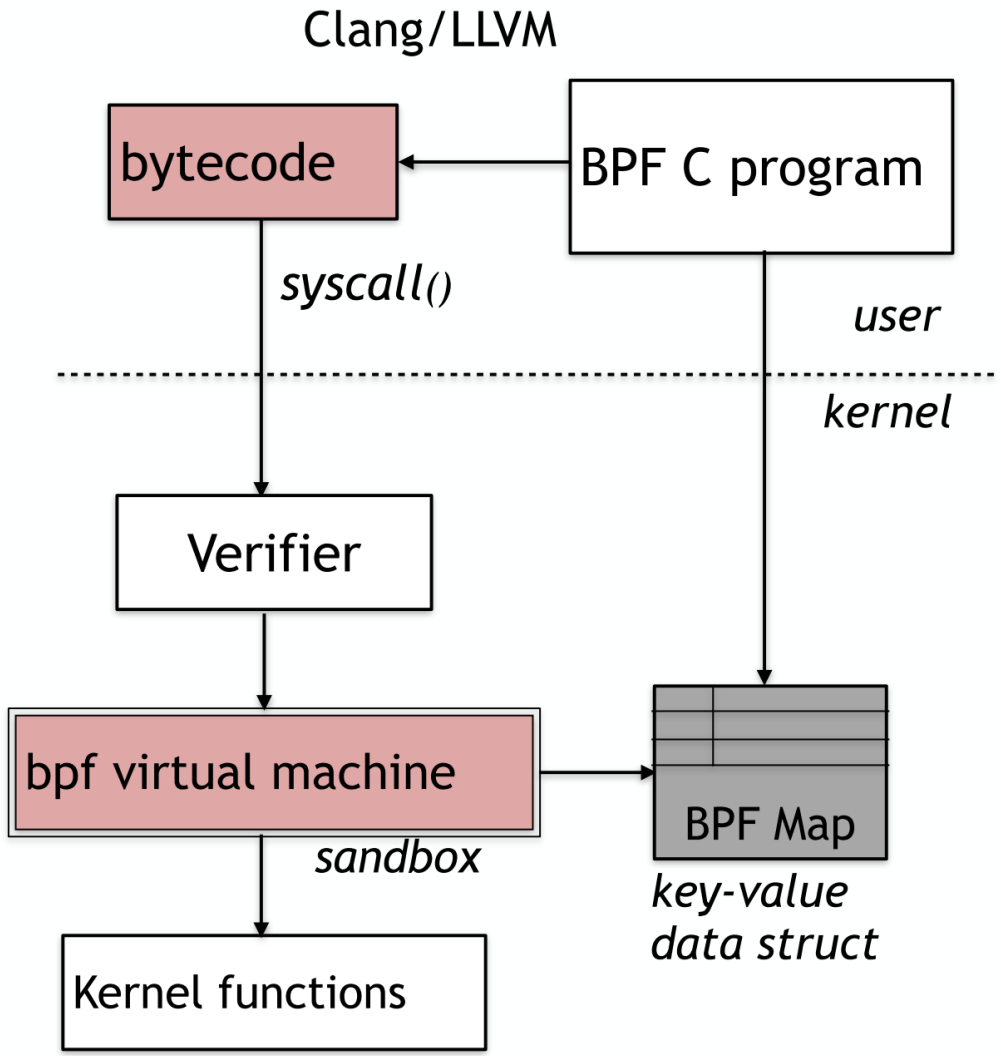
Overview

- Three core components
  - Kernel file system (driver),
  - User library (libExtFUSE)
  - In kernel eBPF virtual machine runtime (VM)
- Fast Path:
  - Only in kernel space
- Slow Path:
  - Cross user space and kernel space



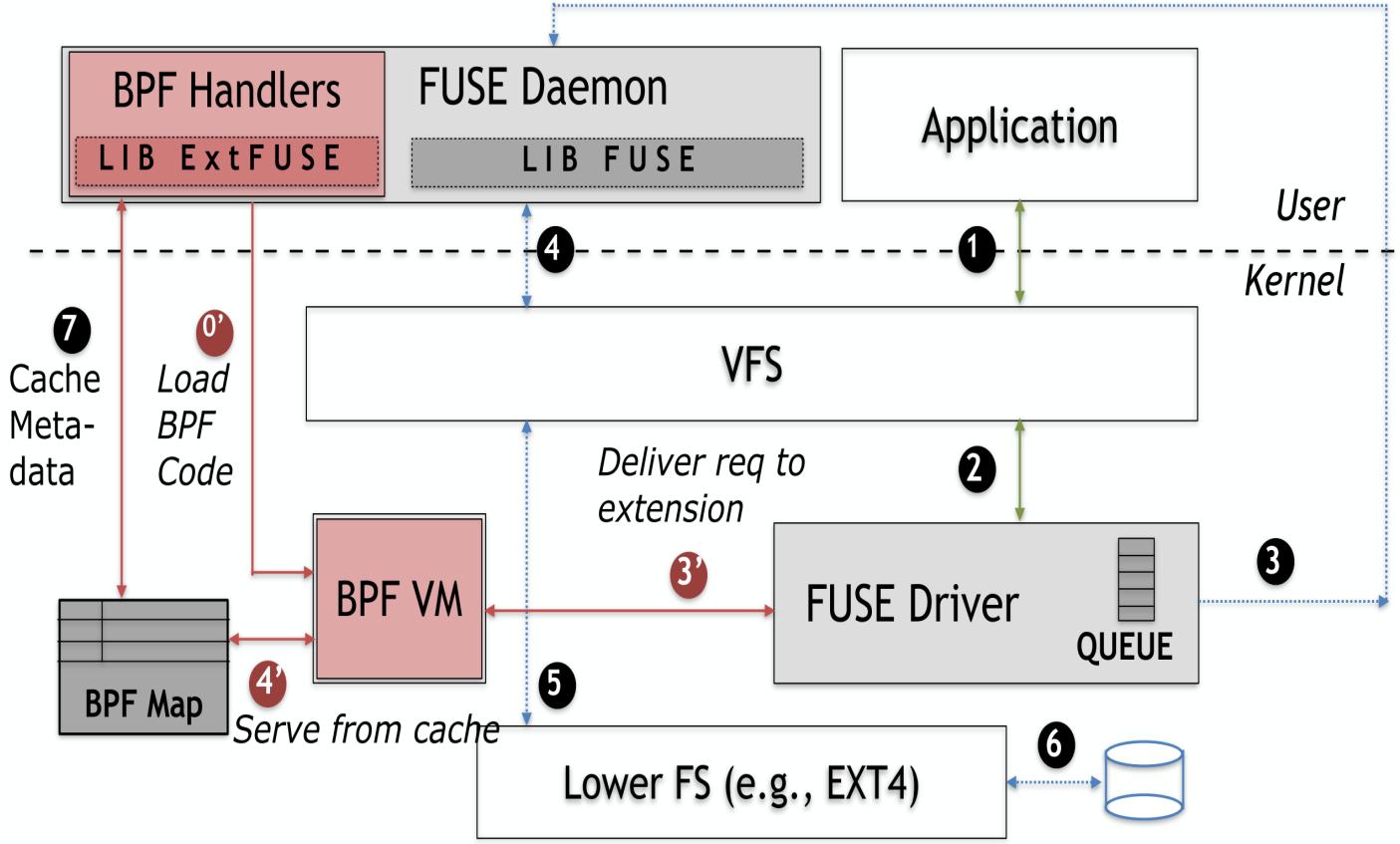
Design

- eBPF (Extended Berkeley Packet Filters)
  - Pseudo machine architecture
  - C code compiled into BPF code
  - Verified and loaded into kernel
  - Executed under VM runtime
  - Evolved as a generic kernel extension framework
  - Used by trace, perf, net subsystems
  - Shared BPF maps with user space



Design

- Register “thin” extensions - handle requests in kernel
  - Avoid user space context switch!
- Share data between FUSE daemon and extensions using BPF maps
  - Cache metadata in the kernel



Design

- Proactively cache/invalidate meta-data in kernel
  - Applies potentially to all FUSE file systems
  - e.g., Gluster readdir ahead results could be cached
- Perform custom filtering or perm checks
  - e.g., Android SDCardFS uid checks in lookup(), open()
- Directly forward I/O requests to lower FS in kernel
  - e.g., install/remove target file descriptor in BPF map

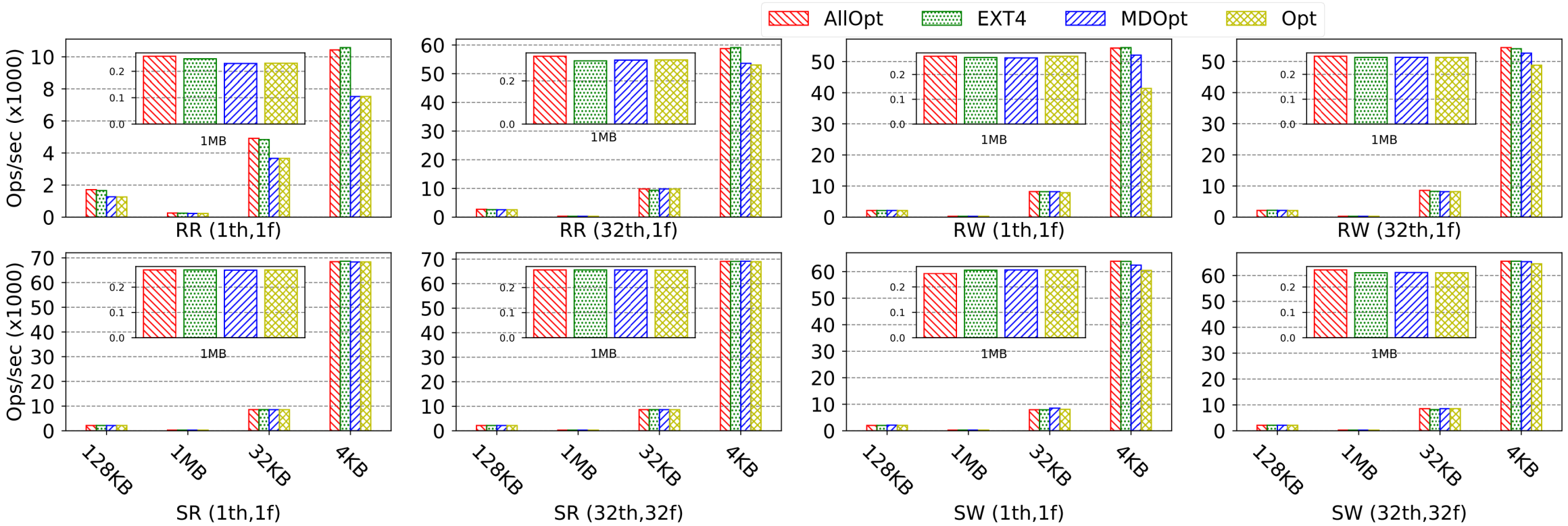
Experimental Setup

- Hardware
  - Samsung 850 EVO 250GB SSD
  - Intel i5-3550 3.3 GHz
  - 16GB RAM
- Software
  - Ubuntu 16.04.3
  - Host file system: ext4

Config	File System	Optimizations
`Opt` [50]	FUSE	128K Writes, Splice, WBCache, MltThrd
`MDOpt`	ExtFUSE	`Opt` + Caches `lookup`, `attrs`, `xattrs`
`AllOpt`	ExtFUSE	`MDOpt` + Pass `R/W` reqs through host FS

Evaluation

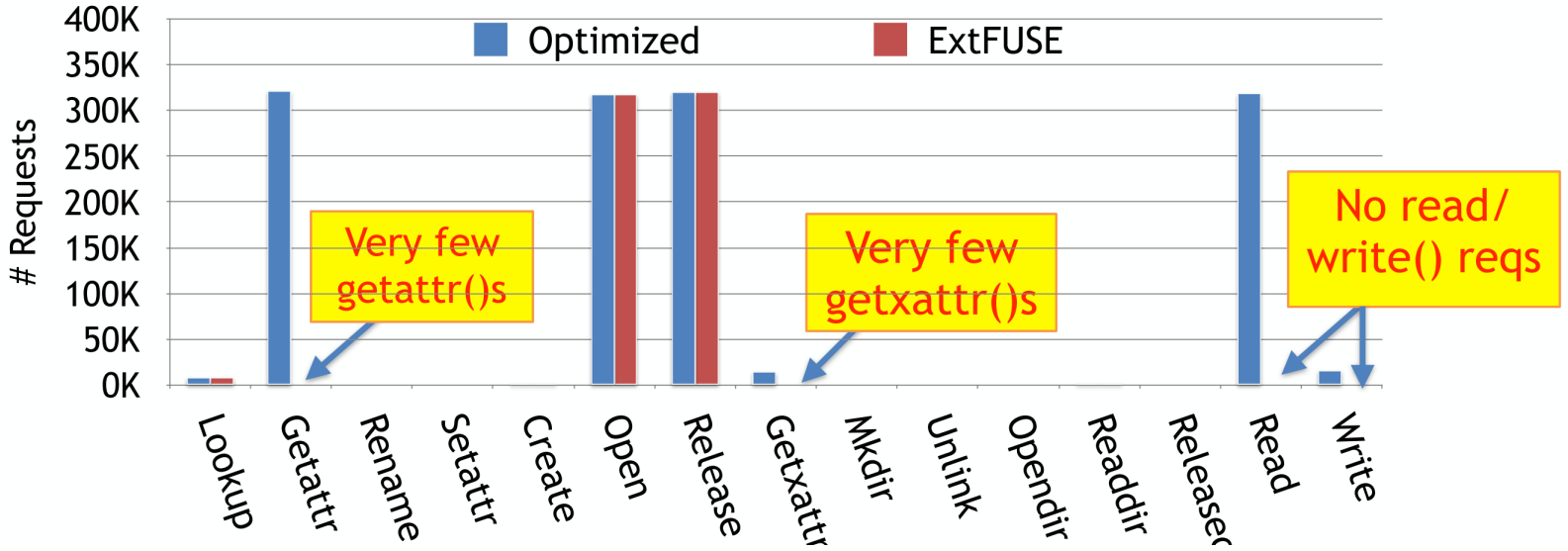
Low overhead compare with Ext4



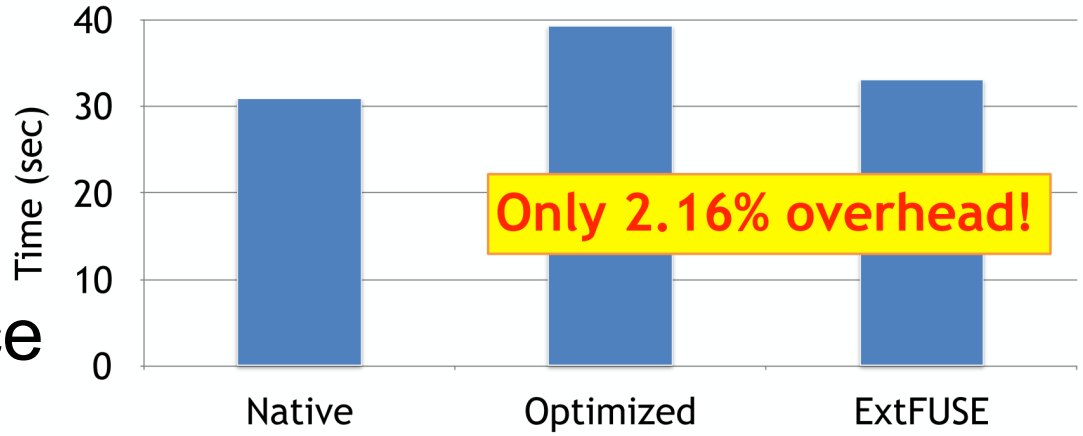
- Throughput(ops/sec) for EXT4 and FUSE/EXTFUSE Stackfs
  - Random Read(RR)/Write(RW), Sequential Read(SR)/Write(SW)
  - Filebench data micro-workloads with IO Sizes between 4KB-1MB.

Evaluation

- "cd linux-4.18; make tinyconfig; make -j4"
- Req received by FUSE



- ExtFUSE Performance



Overhead
Optimized Latency: 28.57%
ExtFUSE Latency: 2.16%
ExtFUSE Memory: 50MB (worst case)
Cached: lookup, attr, xattr
Passthrough: read, write

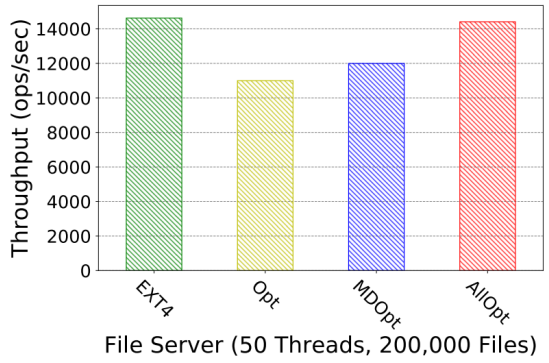
Evaluation

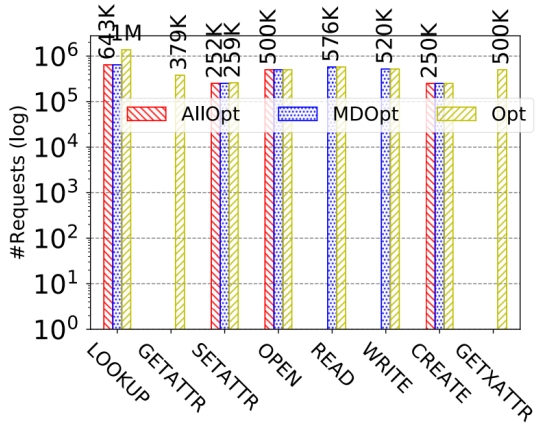
- **Android SDcardFS** App launch latency and peak CPU consumption
  - Default (D)
  - Passthrough (P)

App Stats		CPU (%)		Latency (ms)	
Name	OBB Size	D	P	D	P
Disney Palace Pets 5.1	374MB	20	2.9	2235	1766
Dead Effect 4	1.1GB	20.5	3.2	8895	4579

In passthrough mode, the FUSE driver never forwards read/write requests to user space, but always passes them through the host (EXT4) file system.

- **LoggedFS** performance measured by Filebench FileServer benchmark





Fewer metadata and I/O requests were delivered to user space with EXTFUSE.

Conclusion

- ExtFUSE Features:
  - Cache metadata requests
  - Directly pass I/O requests to lower FS.
  - Insert custom security checks in the kernel.
- Ported four FUSE file systems to ExtFUSE, and show significant performance improvements.

File System	Functionality	Ext Loc
StackFS [50]	No-ops File System	664
BindFS [35]	Mirroring File System	792
Android sdcard [24]	Perm checks & FAT Emu	928
MergerFS [20]	Union File System	686
LoggedFS [16]	Logging File System	748

Lines of code (Loc) of kernel extensions required to adopt ExtFUSE for existing FUSE file systems. Added support for metadata caching as well as R/W passthrough.