Kyu-Jin Cho
(bori19960@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Spring, 2024

FUSE
(Filesystem in Userspace)

Contents

- Userspace Filesystem
- FUSE (Filesystem in Userspace)
- Recent studies
  - ExtFUSE (ATC'19)
  - XFUSE (ATC'21)
  - RFUSE (FAST'24)

Userspace Filesystem

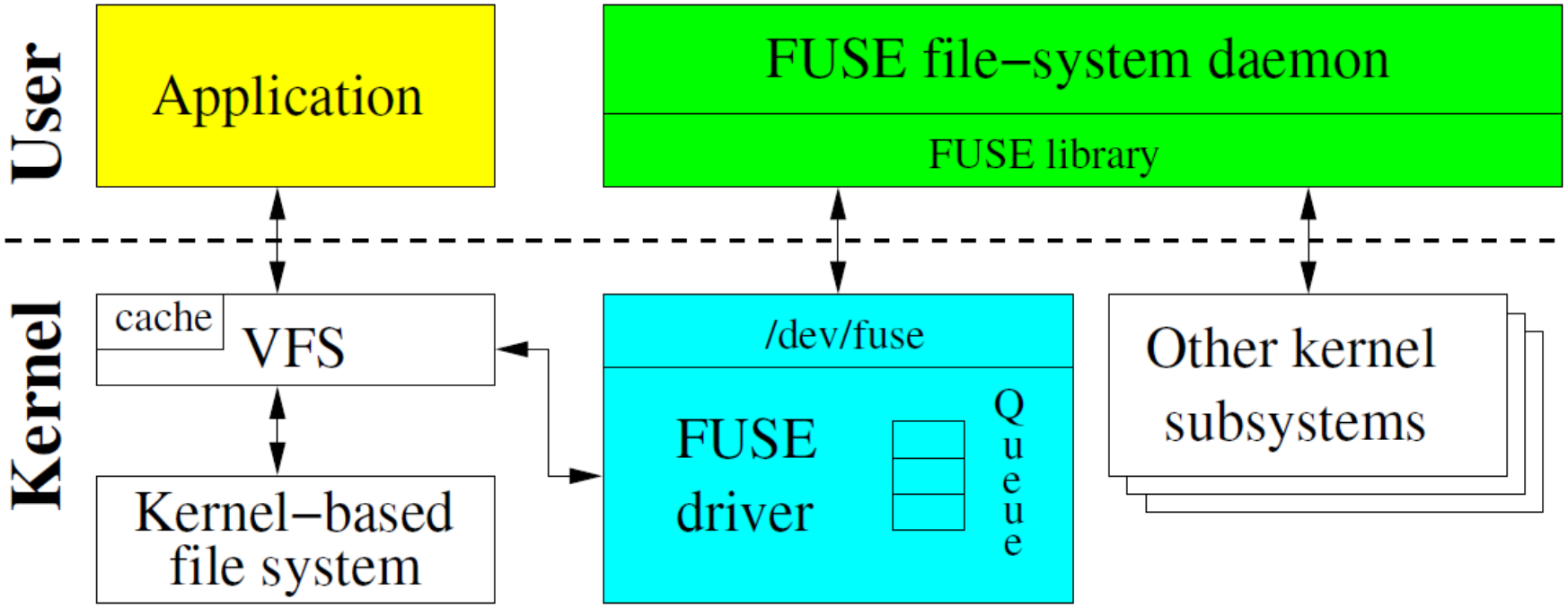
- Transition of filesystem design
  - Traditionally, filesystems were implemented as part of OS kernels
  - As complexity of filesystems grew, filesystems began being developed in userspace

- In-kernel vs. Userspace

In-kernel Filesystem	• Native performance	• Low safety from crash • Complex kernel interface • Hard to add new functionality
Userspace Filesystem	• High safety from crash • Easy to maintain and develop • High portability	• Poor performance

FUSE (Filesystem in Userspace)

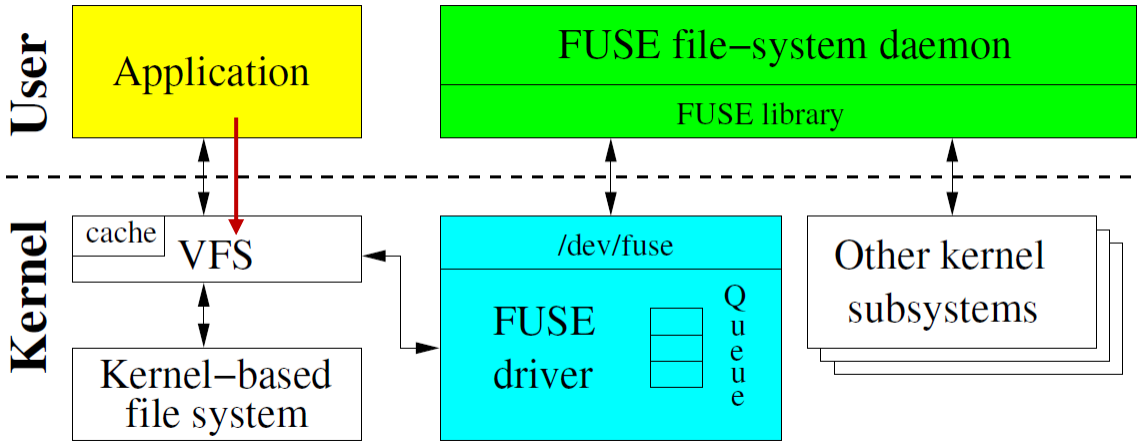
- High-level architecture



* To FUSE or Not to FUSE: Performance of User-Space File Systems (FAST'17)

FUSE Internals

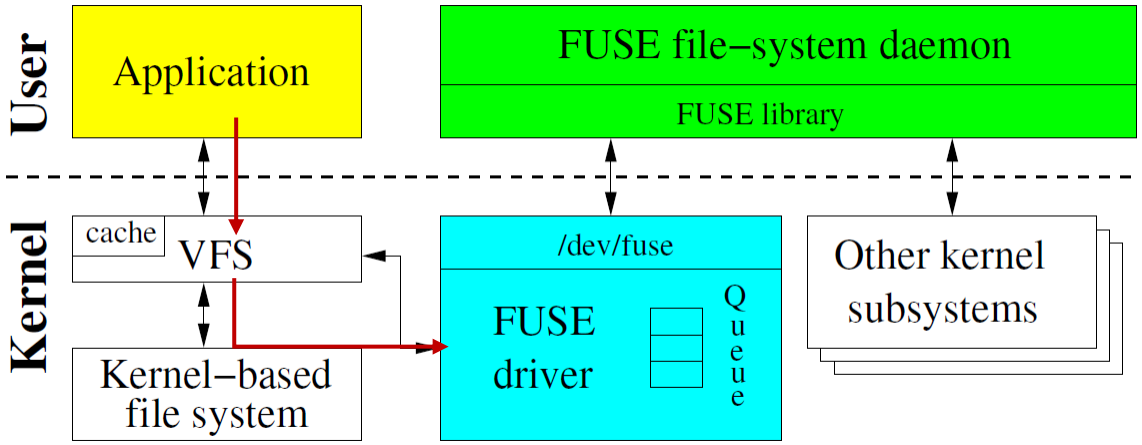
- Request submission



1. User process submits an operation

FUSE Internals

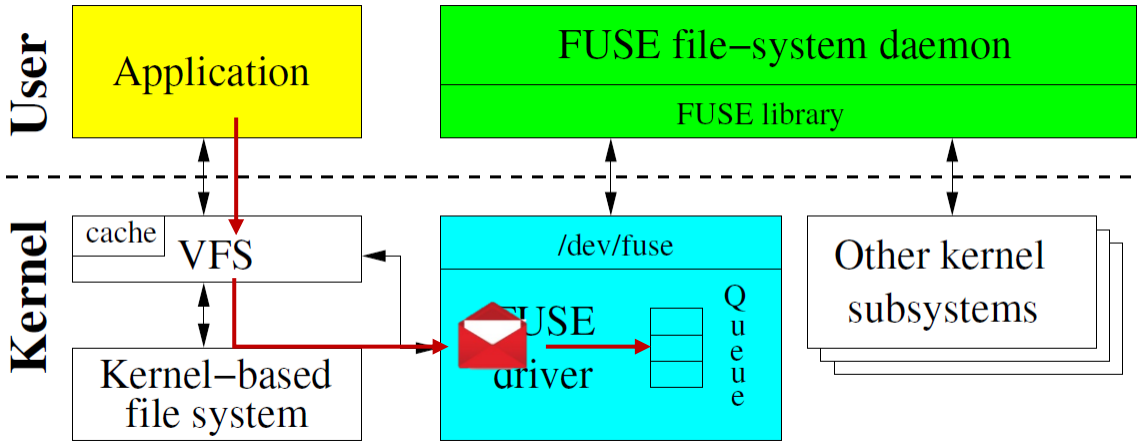
- Request submission



1. User process submits an operation
2. VFS routes the operation to FUSE driver

FUSE Internals

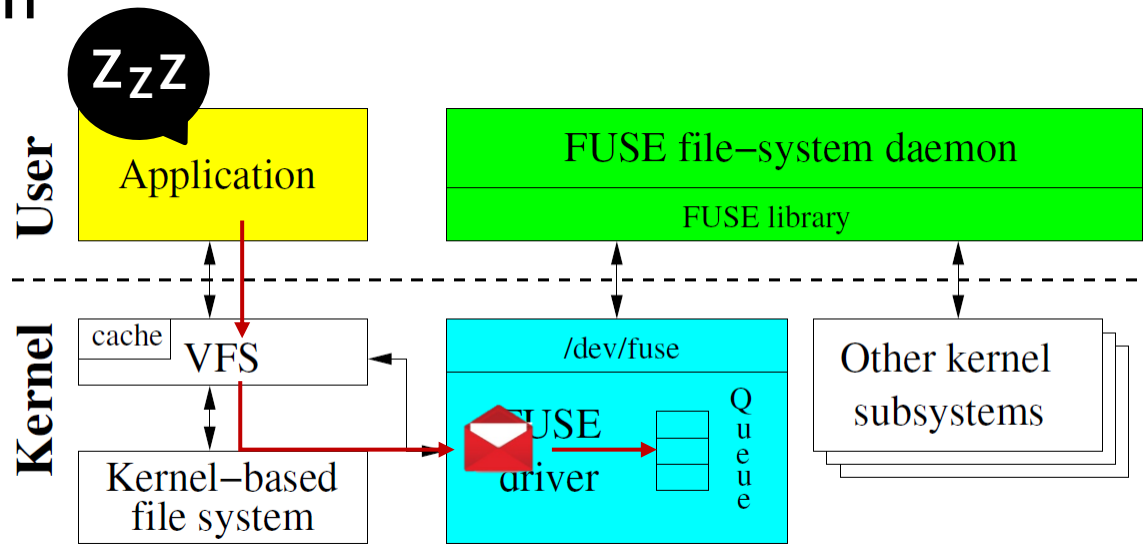
- Request submission



1. User process submits an operation
2. VFS routes the operation to FUSE driver
3. The driver allocates a FUSE request and put it in a FUSE queue

FUSE Internals

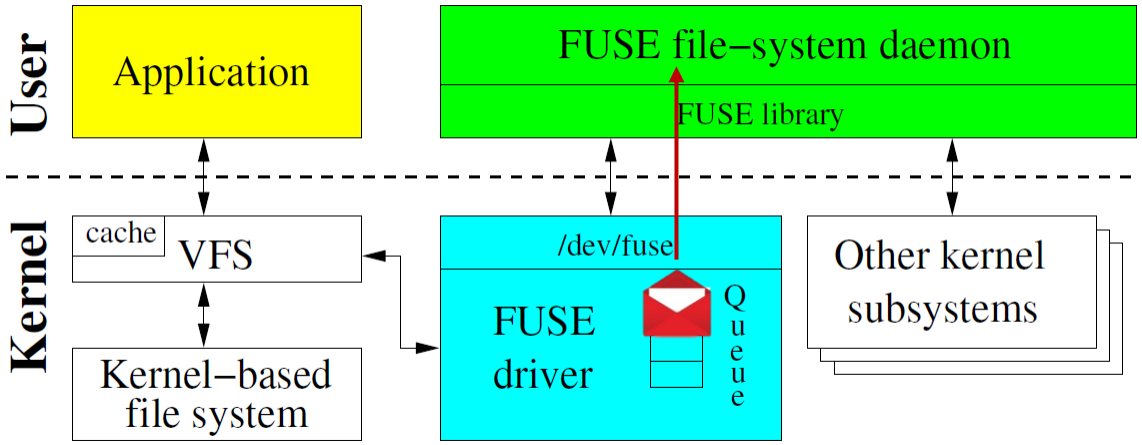
- Request submission



1. User process submits an operation
2. VFS routes the operation to FUSE driver
3. The driver allocates a FUSE request and put it in a FUSE queue
4. The process that submitted the operation is put in a wait state

FUSE Internals

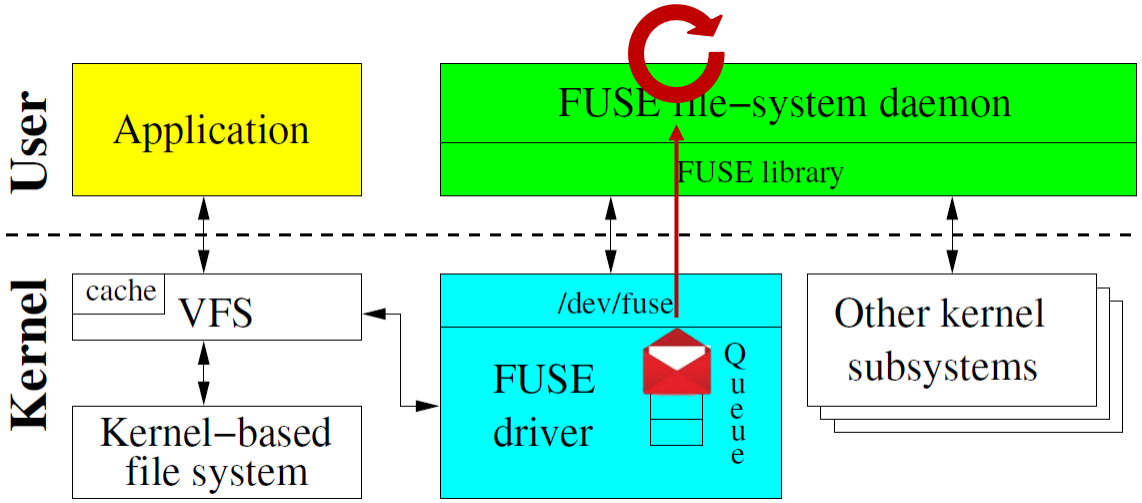
- Request handling & Response



1. FUSE daemon copies the request from the kernel queue by reading */dev/fuse*

FUSE Internals

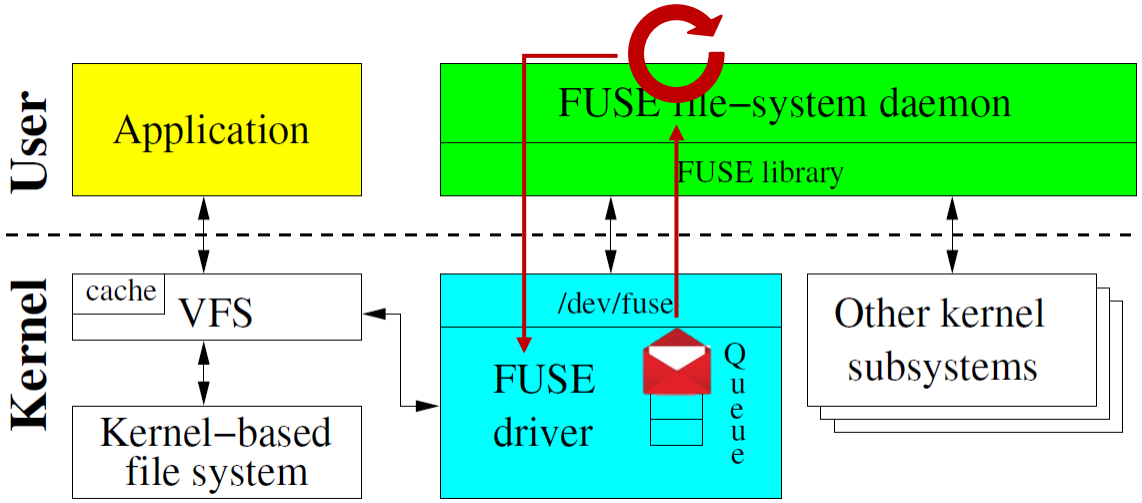
- Request handling & Response



1. FUSE daemon copies the request from the kernel queue by reading */dev/fuse*
2. The daemon processes the request

FUSE Internals

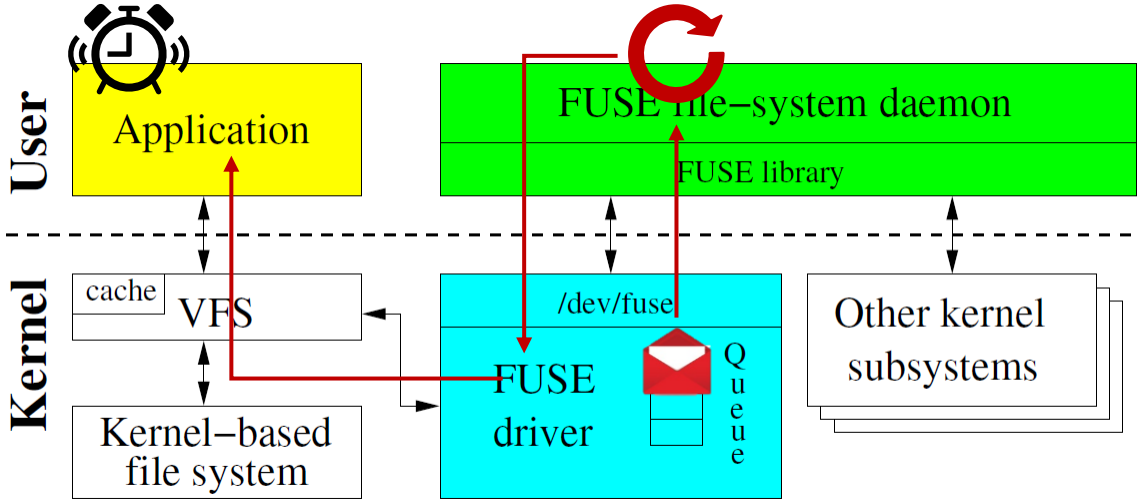
- Request handling & Response



1. FUSE daemon copies the request from the kernel queue by reading */dev/fuse*
2. The daemon processes the request
3. The daemon write the response back to */dev/fuse*

FUSE Internals

- Request handling & Response



1. FUSE daemon copies the request from the kernel queue by reading */dev/fuse*
2. The daemon processes the request
3. The daemon write the response back to */dev/fuse*
4. The driver marks the request as completed and wakes up the user process

FUSE Internals

- User-Kernel Protocol
  - Kernel and user use identical header files for interoperability
  - Kernel: (*include/uapi/linux/fuse.h*)
  - Libfuse: (*include/fuse_kernel.h*)

```
struct fuse_in_header {
        uint32_t        len;
        uint32_t        opcode;
        uint64_t        unique;
        uint64_t        nodeid;
        uint32_t        uid;
        uint32_t        gid;
        uint32_t        pid;
        uint32_t        padding;
};

struct fuse_out_header {
        uint32_t        len;
        int32_t                 error;
        uint64_t        unique;
};
```

<Common header>

```
struct fuse_write_in {
        uint64_t        fh;
        uint64_t        offset;
        uint32_t        size;
        uint32_t        write_flags;
        uint64_t        lock_owner;
        uint32_t        flags;
        uint32_t        padding;
};

struct fuse_write_out {
        uint32_t        size;
        uint32_t        padding;
};
```

<Operation-specific header>

FUSE Internals

- Request types

Group (#)	Request Types
Special (3)	**INIT**, **DESTROY**, **INTERRUPT**
Metadata (14)	**LOOKUP**, **FORGET**, **BATCH_FORGET**, CREATE, UNLINK, LINK, RENAME, RENAME2, **OPEN**, **RELEASE**, STATFS, FSYNC, **FLUSH**, ACCESS
Data (2)	READ, WRITE
Attributes (2)	GETATTR, SETATTR
Extended Attributes (4)	SETXATTR, GETXATTR, LISTXATTR, REMOVEXATTR
Symlinks (2)	SYMLINK, READLINK
Directory (7)	MKDIR, RMDIR, OPENDIR, **RELEASEDIR**, READDIR, **READDIRPLUS**, FSYNCDIR
Locking (3)	GETLK, SETLK, SETLKW
Misc (6)	BMAP, FALLOCATE, MKNOD, IOCTL, POLL, NOTIFY_REPLY

FUSE Internals

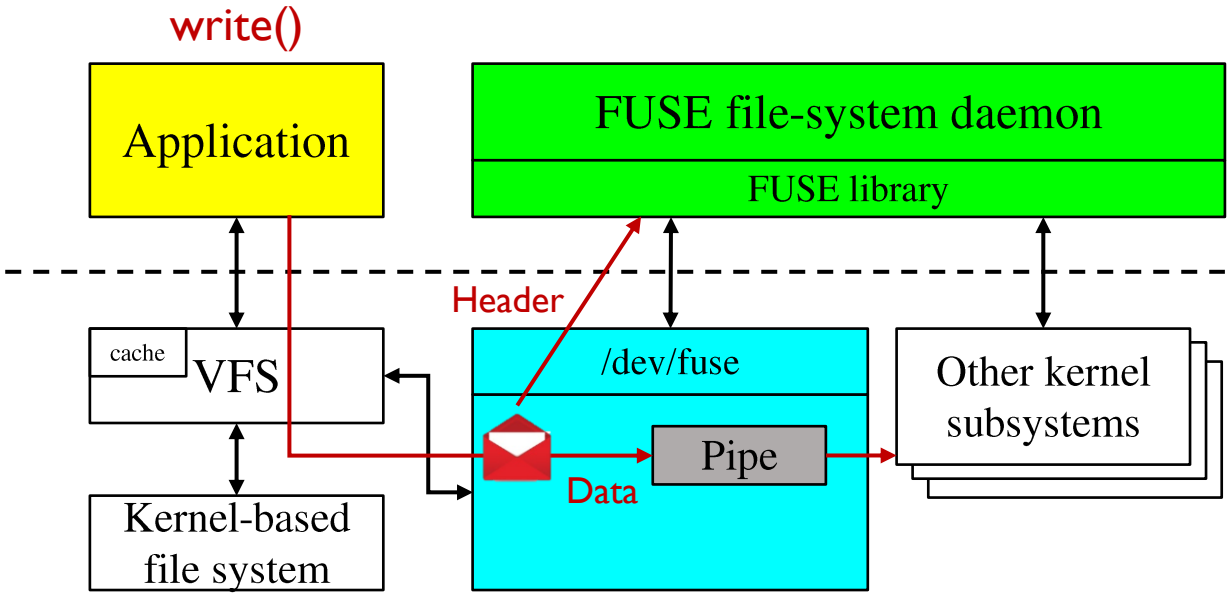
- Request types
  - INIT
    - Sent by kernel during mounting process
    - Check protocol version
    - Set mutually supported capabilities and mount options
  - DESTROY
    - Sent by kernel during unmouting process
    - FUSE daemon is expected to perform all cleanups

FUSE Internals

- Request types
  - INTERRUPT
    - Sent by the kernel if any requests that were previously passed to the daemon are no logger needed (e.g. when a user process blocked on READ is terminated)
    - Each request has a unique sequence number which INTERRUPT used to identify victim requests
  - FORGET
    - Sent by kernel when an inode is removed from the kernel dcache
    - The daemon might decide to deallocate any corresponding data structures

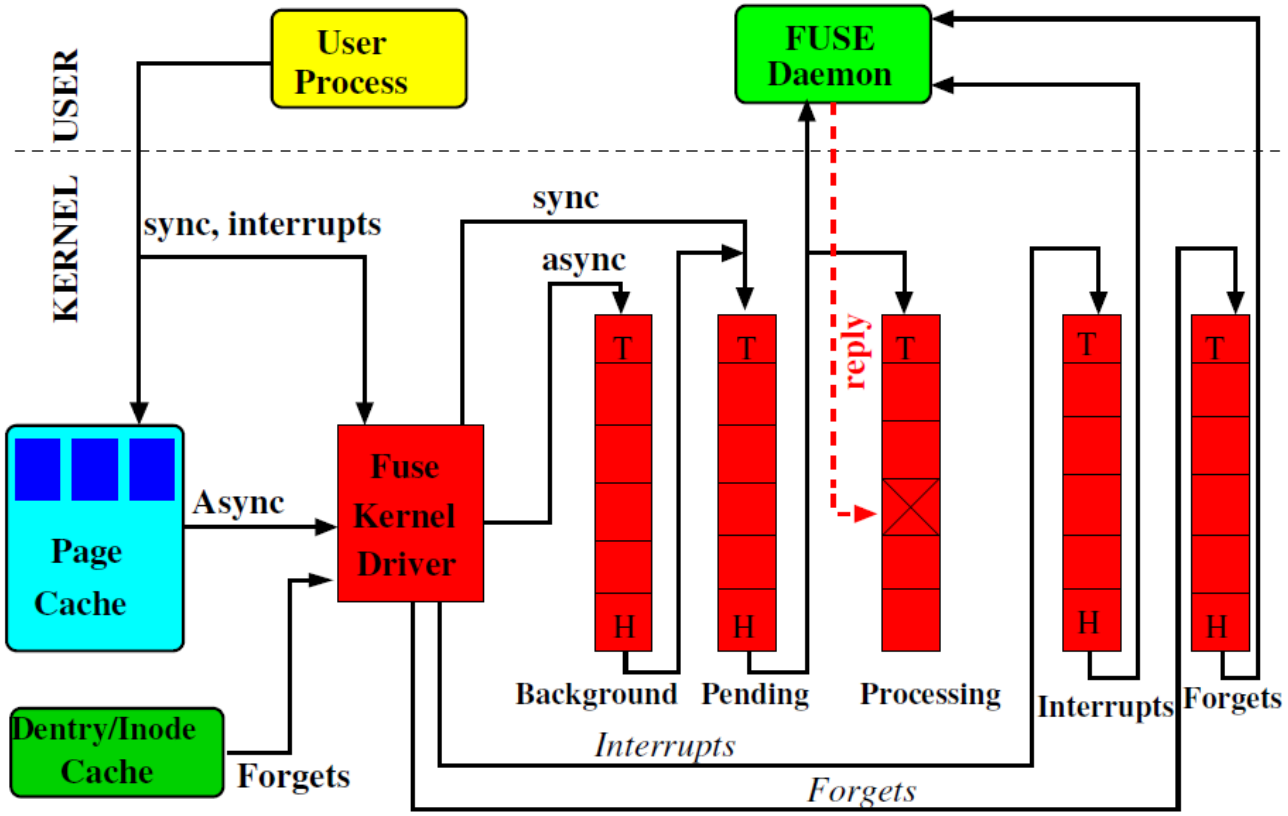
FUSE Internals

- Splicing
  - Prevent a memory copy between the kernel and userspace
  - Useful for stackable filesystems
  - However, memory copying is always performed for the header



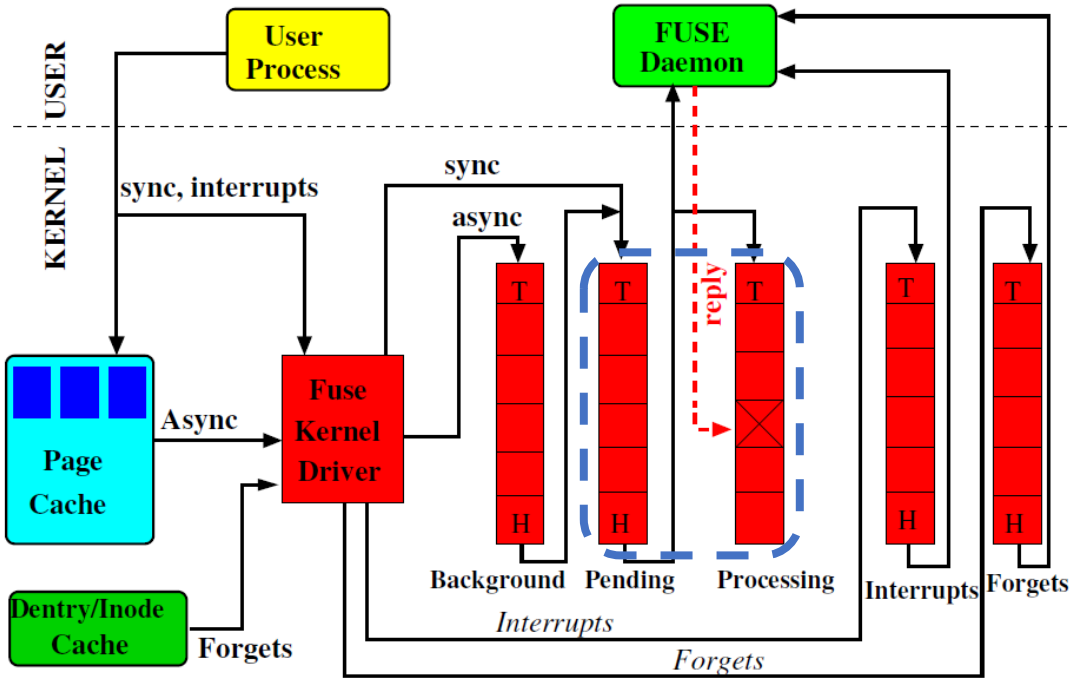
FUSE Internals

- Queues



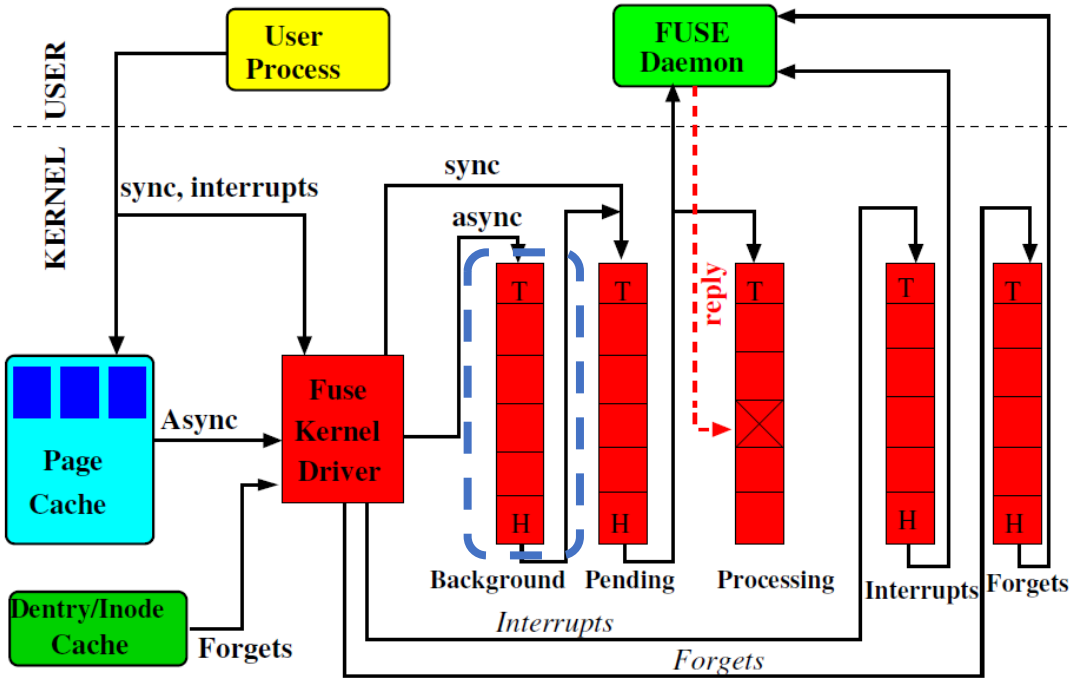
FUSE Internals

- Queues
  - Pending queue
    - Staging submitted request
  - Processing queue
    - The oldest pending request is sent to the FUSE daemon and simultaneously moved to it
  - When the demon replies to the request, it is removed from processing queue



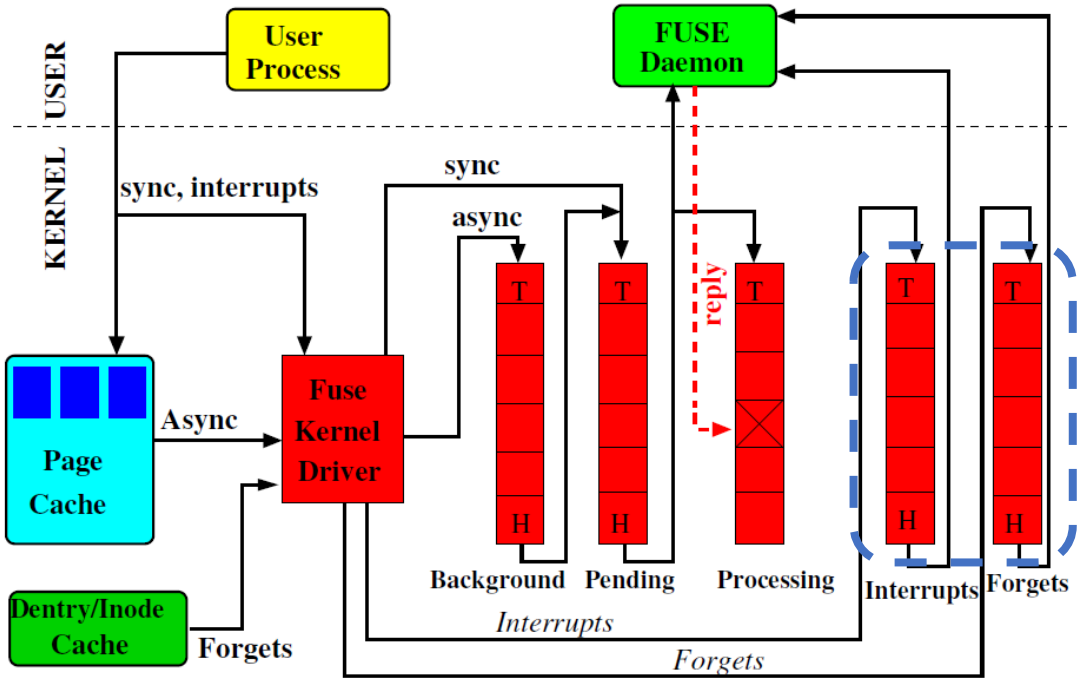
FUSE Internals

- Queues
  - Background queue
    - Staging asynchronous requests (E.g. init, release, write-back, readahead)
    - Limit the number of async request simultaneously residing in the pending queue (default: 12)
    - Limit the delay caused to important synchronous requests by bursts of background request



FUSE Internals

- Queues
  - Interrupts queue
    - For assigning high priority INTERRUPT requests
  - Forgets queue
    - For FORGET requests to differentiate them from non-forget requests
    - To prevent FUSE daemon to be stuck by bursty FORGET request



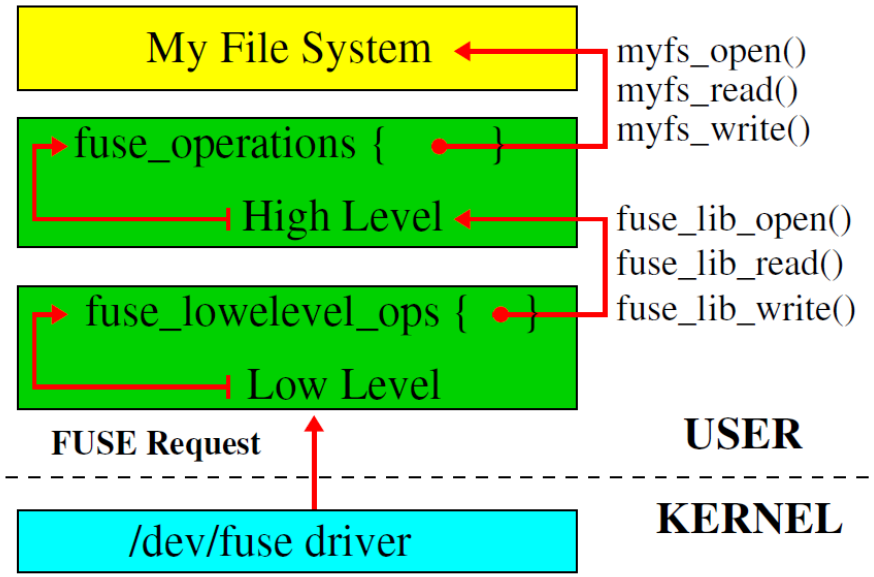
FUSE Internals

- Library and API Levels
  - Low-level API
    - Flexibility
    - Communicate with the kernel directly
    - Take fuse request as an argument
    - Need <fuse_inode - path> mapping

```
void (*open) (fuse_req_t req, fuse_ino_t ino,
              struct fuse_file_info *fi);
```

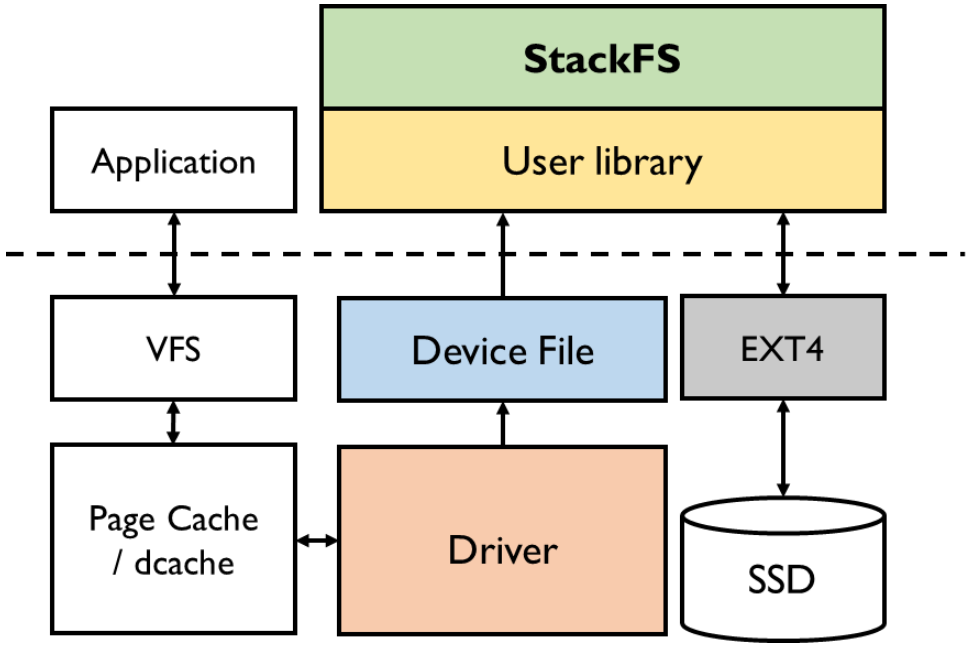
  - High-level API
    - Development ease
    - Skip the implementation of fuse_inode-path mapping
    - Not need forget() method

```
int (*open) (const char *, struct fuse_file_info *);
```



FUSE Example

- Stackable Filesystem (StackFS)
  - A stackable filesystem that forwards incoming filesystem operations to an underlying in-kernel filesystem (e.g. EXT4, F2FS, etc.)



FUSE Example

- Parse fuse options and call "*fuse_main()*" function

```
int main(int argc, char *argv[])
{
    enum { MAX_ARGS = 10 };
    int i,new_argc;
    char *new_argv[MAX_ARGS];

    umask(0);
            /* Process the "--plus" option apart */
    for (i=0, new_argc=0; (i<argc) && (new_argc<MAX_ARGS); i++) {
        if (!strcmp(argv[i], "--plus")) {
            fill_dir_plus = FUSE_FILL_DIR_PLUS;
        } else {
            new_argv[new_argc++] = argv[i];
        }
    }
    return fuse_main(new_argc, new_argv, &xmp_oper, NULL);
}
```

FUSE Example

- Parse fuse options and call "*fuse_main()*" function

```
int main(int argc, char *argv[])
{
    enum { MAX_ARGS = 10 };
    int i,new_argc;
    char *new_argv[MAX_ARGS];

    umask(0);
            /* Process the "--plus" option apart */
    for (i=0, new_argc=0; (i<argc) && (new_argc<MAX_ARGS); i++) {
        if (!strcmp(argv[i], "--plus")) {
            fill_dir_plus = FUSE_FILL_DIR_PLUS;
        } else {
            new_argv[new_argc++] = argv[i];
        }
    }
    return fuse_main(new_argc, new_argv, &xmp_oper, NULL);
}
```

FUSE Example

- Parse fuse options and call "*fuse_main()*" function

```
int main(int argc, char *argv[])
{
    enum { MAX_ARGS = 10 };
    int i,new_argc;
    char *new_argv[MAX_ARGS];

    umask(0);
            /* Process the "--plus" option apart */
    for (i=0, new_argc=0; (i<argc) && (new_argc<MAX_ARGS); i++) {
        if (!strcmp(argv[i], "--plus")) {
            fill_dir_plus = FUSE_FILL_DIR_PLUS;
        } else {
            new_argv[new_argc++] = argv[i];
        }
    }
    return fuse_main(new_argc, new_argv, &xmp_oper, NULL);
}
```

FUSE Example

- Declare "*struct fuse_operations*"

```c
int main(int argc, char *argv[])
{
    enum { MAX_ARGS = 10 };
    int i,new_argc;
    char *new_argv[MAX_ARGS];

    umask(0);
            /* Process the "--plus" option apart */
    for (i=0, new_argc=0; (i<argc) && (new_argc<MAX_ARGS); i++) {
        if (!strcmp(argv[i], "--plus")) {
            fill_dir_plus = FUSE_FILL_DIR_PLUS;
        } else {
            new_argv[new_argc++] = argv[i];
        }
    }
    return fuse_main(new_argc, new_argv, &xmp_oper, NULL);
}
```

```c
static const struct fuse_operations xmp_oper = {
        .init           = xmp_init,
        .getattr        = xmp_getattr,
        .access         = xmp_access,
        .readlink       = xmp_readlink,
        .readdir        = xmp_readdir,
        .mknod          = xmp_mknod,
        .mkdir          = xmp_mkdir,
        .symlink        = xmp_symlink,
        .unlink         = xmp_unlink,
        .rmdir          = xmp_rmdir,
        .rename         = xmp_rename,
        ...
```

FUSE Example

- Declare "*struct fuse_operations*"

```c
int main(int argc, char *argv[])
{
    enum { MAX_ARGS = 10 };
    int i,new_argc;
    char *new_argv[MAX_ARGS];

    umask(0);
            /* Process the "--plus" option apart */
    for (i=0, new_argc=0; (i<argc) && (new_argc<MAX_ARGS); i++) {
        if (!strcmp(argv[i], "--plus")) {
            fill_dir_plus = FUSE_FILL_DIR_PLUS;
        } else {
            new_argv[new_argc++] = argv[i];
        }
    }
    return fuse_main(new_argc, new_argv, &xmp_oper, NULL);
}
```

```c
static const struct fuse_operations xmp_oper = {
        .init           = xmp_init,
        .getattr        = xmp_getattr,
        .access         = xmp_access,
        .readlink       = xmp_readlink,
        .readdir        = xmp_readdir,
        .mknod          = xmp_mknod,
        .mkdir          = xmp_mkdir,
        .symlink        = xmp_symlink,
        .unlink         = xmp_unlink,
        .rmdir          = xmp_rmdir,
        .rename         = xmp_rename,
        ...
```

FUSE Example

- Implement filesystem operations

```
static void *xmp_init(struct fuse_conn_info *conn,
                      struct fuse_config *cfg)
{
    (void) conn;
    cfg->use_ino = 1;

    cfg->entry_timeout = 0;
    cfg->attr_timeout = 0;
    cfg->negative_timeout = 0;

    return NULL;
}
```

FUSE Example

- Implement filesystem operations

```
static void *xmp_init(struct fuse_conn_info *conn,
                      struct fuse_config *cfg)
{
    (void) conn;
    cfg->use_ino = 1;

    cfg->entry_timeout = 0;
    cfg->attr_timeout = 0;
    cfg->negative_timeout = 0;

    return NULL;
}
```

FUSE Example

- Implement filesystem operations

```
static void *xmp_init(struct fuse_conn_info *conn,
                      struct fuse_config *cfg)
{
    (void) conn;
    cfg->use_ino = 1;

    cfg->entry_timeout = 0;
    cfg->attr_timeout = 0;
    cfg->negative_timeout = 0;

    return NULL;
}
```

```
static int xmp_getattr(const char *path, struct stat *stbuf,
                       struct fuse_file_info *fi)
{
    (void) fi;
    int res;

    res = lstat(path, stbuf);
    if (res == -1)
        return -errno;

    return 0;
}
```

FUSE Example

- Implement filesystem operations

```
static void *xmp_init(struct fuse_conn_info *conn,
                      struct fuse_config *cfg)
{
    (void) conn;
    cfg->use_ino = 1;

    cfg->entry_timeout = 0;
    cfg->attr_timeout = 0;
    cfg->negative_timeout = 0;

    return NULL;
}
```

```
static int xmp_getattr(const char *path, struct stat *stbuf,
                       struct fuse_file_info *fi)
{
    (void) fi;
    int res;

    res = lstat(path, stbuf);
    if (res == -1)
        return -errno;

    return 0;
}
```

FUSE Example

- Document: https://libfuse.github.io/doxygen/index.html

libfuse

Main Page | Data Structures | Files

Data Fields

fuse_operations Struct Reference

#include <fuse.h>

Data Fields

int(* getattr)(const char *, struct stat *, struct fuse_file_info *fi)
int(* readlink)(const char *, char *, size_t)
int(* mknod)(const char *, mode_t, dev_t)
int(* mkdir)(const char *, mode_t)
int(* unlink)(const char *)
int(* rmdir)(const char *)
int(* symlink)(const char *, const char *)
int(* rename)(const char *, const char *, unsigned int flags)
int(* link)(const char *, const char *)
int(* chmod)(const char *, mode_t, struct fuse_file_info *fi)
int(* chown)(const char *, uid_t, gid_t, struct fuse_file_info *fi)
int(* truncate)(const char *, off_t, struct fuse_file_info *fi)
int(* open)(const char *, struct fuse_file_info *)
int(* read)(const char *, char *, size_t, off_t, struct fuse_file_info *)
int(* write)(const char *, const char *, size_t, off_t, struct fuse_file_info *)

FUSE Example

- Document: https://libfuse.github.io/doxygen/index.html

◆ init

void *(* fuse_operations::init) (struct **fuse_conn_info** *conn, struct **fuse_config** *cfg)

Initialize filesystem

The return value will passed in the `private_data` field of `struct fuse_context` to all file operations, and as a parameter to the **destroy()** method. It overrides the initial value provided to **fuse_main()** / fuse_new().

Definition at line **616** of file **fuse.h**.

◆ getattr

int(* fuse_operations::getattr) (const char *, struct stat *, struct **fuse_file_info** *fi)

Get file attributes.

Similar to stat(). The 'st_dev' and 'st_blksize' fields are ignored. The 'st_ino' field is ignored except if the 'use_ino' mount option is given. In that case it is passed to userspace, but libfuse and the kernel will still assign a different inode for internal use (called the "nodeid").

`fi` will always be NULL if the file is not currently open, but may also be NULL if the file is open.

Definition at line **336** of file **fuse.h**.

* To FUSE or Not to FUSE: Performance of User-Space File Systems (FAST'17)
* StackFS-Base: Single-threaded FUSE daemon and Copy-based I/O without write-back cache
* SOpt : Multi-threaded FUSE daemon and Splicing I/O

FUSE Overhead

- Sequential I/O on StackFS

Workload	I/O Size (KB)	HDD Results			SSD Results		
		EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)	EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)
seq-wr-32th-32f	4	34370	- 2.5^{+}	+ 0.1^{+}	32921	+ 0.05^{+}	+ 0.2^{+}
	32	4296	- 2.7^{+}	+ 0.0^{+}	4115	+ 0.1^{+}	+ 0.1^{+}
	128	1075	- 2.6^{+}	- 0.02^{+}	1029	- 0.04^{+}	+ 0.2^{+}
	1024	134	- 2.4^{+}	- 0.18^{+}	129	- 0.1^{+}	+ 0.2^{+}
seq-rd-32th-32f	4	11141	- 36.9$^{\#}$	- 26.9$^{\#}$	32855	- 0.1^{+}	- 0.16^{+}
	32	1491	- 41.5$^{\#}$	- 30.3$^{\#}$	4202	- 0.1^{+}	- 1.8^{+}
	128	371	- 41.3$^{\#}$	- 29.8$^{\#}$	1051	- 0.1^{+}	- 0.2^{+}
	1024	46	- 41.0$^{\#}$	- 28.3$^{\#}$	131	- 0.03^{+}	- 2.1^{+}

FUSE Overhead

* To FUSE or Not to FUSE: Performance of User-Space File Systems (FAST'17)
* StackFS-Base: Single-threaded FUSE daemon and Copy-based I/O without write-back cache
* SOpt : Multi-threaded FUSE daemon and Splicing I/O

- Random I/O on StackFS

Workload	I/O Size (KB)	HDD Results			SSD Results		
		EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)	EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)
rnd-wr-32th-1f	4	1073	- 0.9^{+}	- 1.8^{+}	16213	- 0.7^{+}	- 26.6$^{\#}$
	32	705	+ 0.1^{+}	- 0.7^{+}	4103	- 2.2^{+}	- 13.0*
	128	358	+ 0.3^{+}	- 1.1^{+}	1031	- 0.1^{+}	+ 0.03^{+}
	1024	79	+ 0.1^{+}	- 0.3^{+}	128	+ 0.9^{+}	- 0.3^{+}
rnd-rd-32th-1f	4	572	- 60.4$^{!}$	-23.2*	24998	- 82.5$^{!}$	-27.6$^{\#}$
	32	504	- 56.2$^{!}$	-17.2*	4273	- 55.7$^{!}$	-1.9^{+}
	128	278	- 34.4$^{\#}$	-11.4*	1123	- 29.1$^{\#}$	-2.6^{+}
	1024	41	- 37.0$^{\#}$	-15.0*	126	- 12.2*	-1.9^{+}

* To FUSE or Not to FUSE: Performance of User-Space File Systems (FAST'17)
* StackFS-Base: Single-threaded FUSE daemon and Copy-based I/O without write-back cache
* SOpt : Multi-threaded FUSE daemon and Splicing I/O

FUSE Overhead

- Metadata operations on StackFS

Workload	I/O Size (KB)	HDD Results			SSD Results		
		EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)	EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)
files-cr-1th	4	30211	- 57!	- 81.0!	35361	- 62.2!	- 83.3!
files-cr-32th	4	36590	- 50.2!	- 54.9!	46688	- 57.6!	- 62.6!
files-rd-1th	4	645	+ 0.0+	- 10.6*	8055	- 25.0*	- 60.3!
files-rd-32th	4	1263	- 50.5!	-4.5+	25341	- 74.1!	-33.0#
files-del-1th	-	1105	- 4.0+	- 10.2*	7391	- 31.6#	- 60.7!
files-del-32th	-	1109	- 2.8+	- 6.9*	8563	- 42.9#	- 52.6!

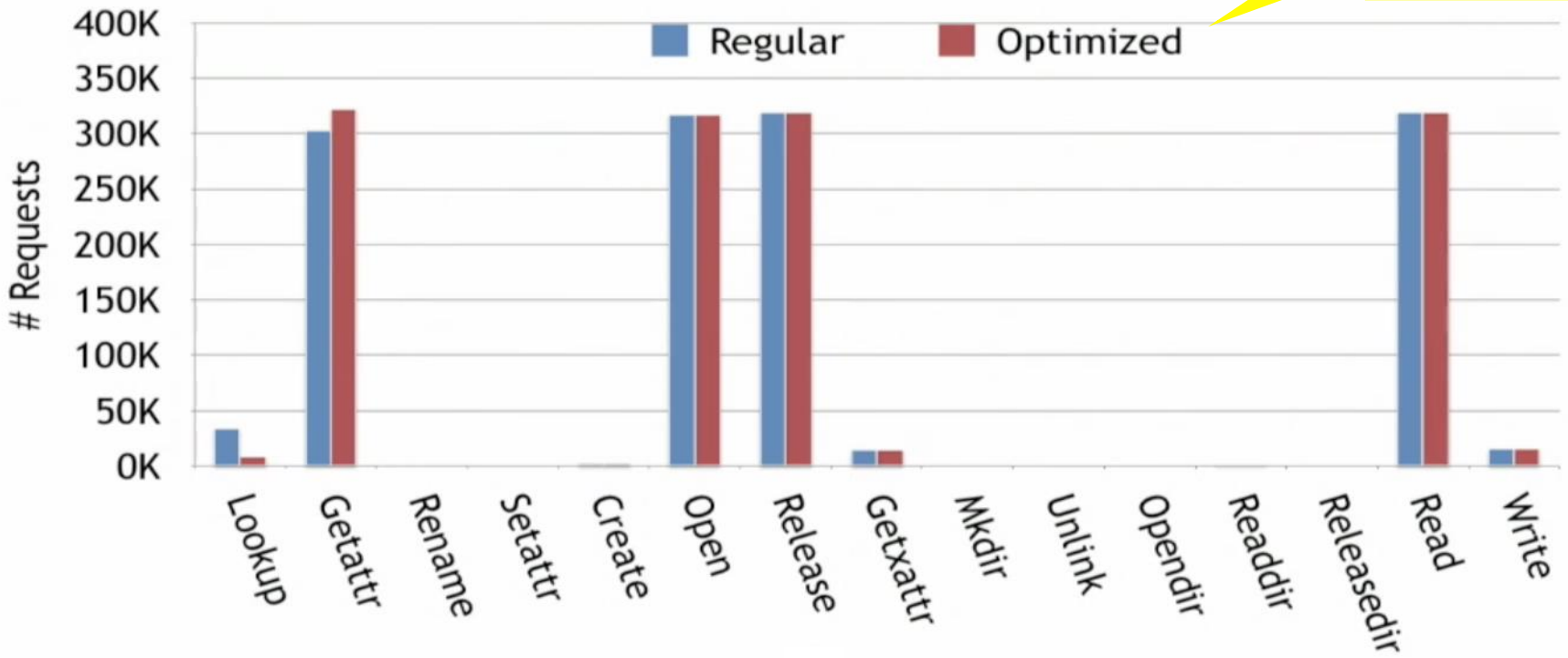
ExtFUSE

* Extension Framework for File Systems in User space (ATC'19)

Motivation

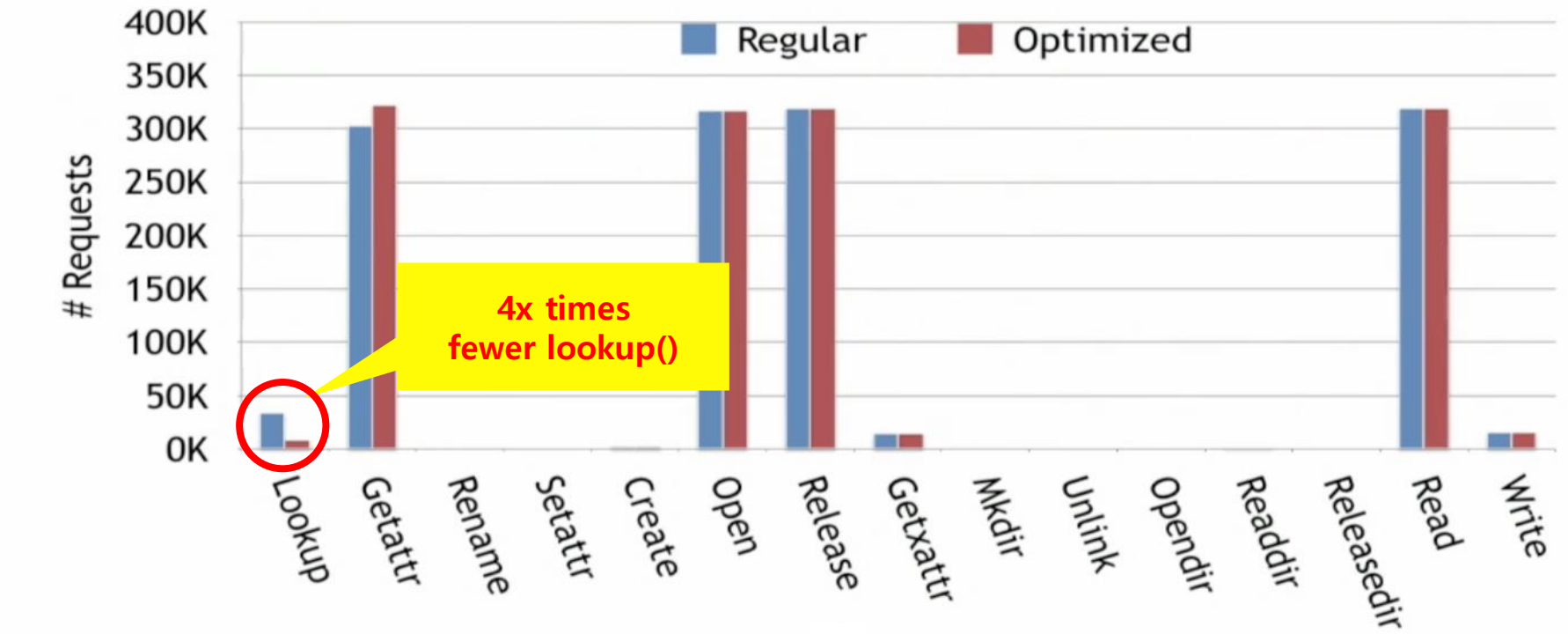
- FUSE performance
  - "cd linux-4.18; make tinyconfig; make –j4
    - # of Request received by FUSE

with splicing and system wide VFS cache



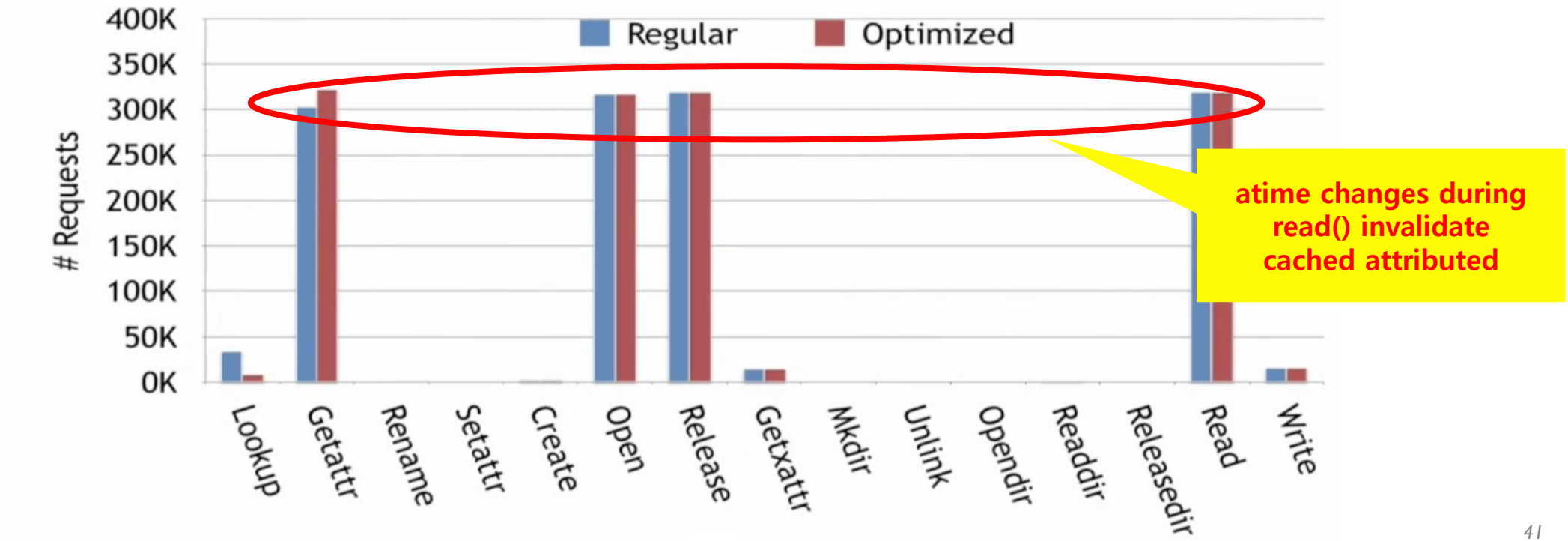
Motivation

- FUSE performance
  - "cd linux-4.18; make tinyconfig; make –j4
    - # of Request received by FUSE



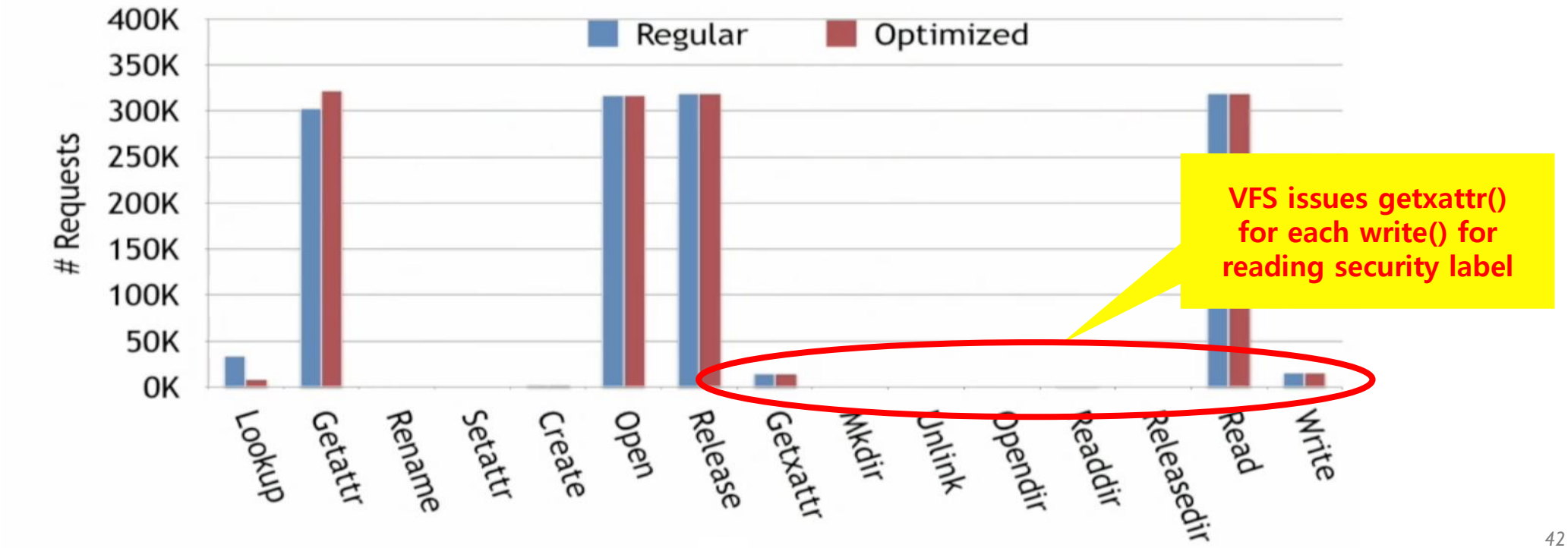
Motivation

- FUSE performance
  - "cd linux-4.18; make tinyconfig; make –j4
    - # of Request received by FUSE



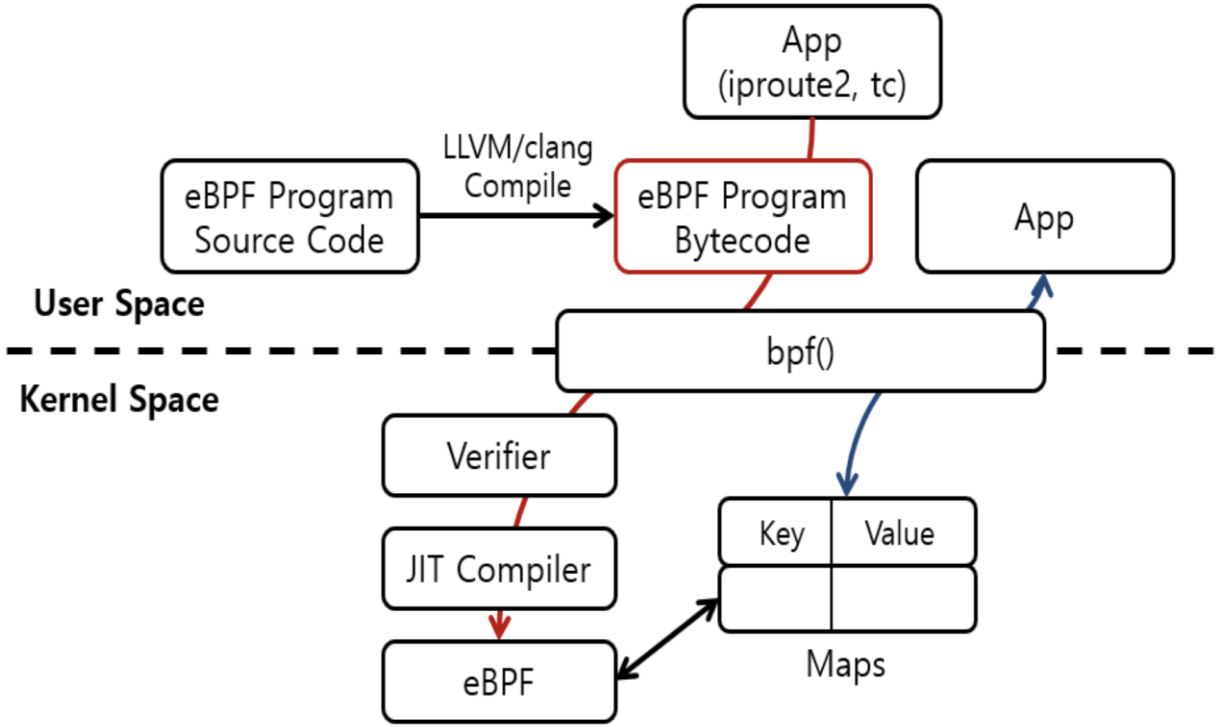
Motivation

- FUSE performance
  - "cd linux-4.18; make tinyconfig; make –j4
    - # of Request received by FUSE



Background

- eBPF (extended Berkely Packet Filter)
  - Pseudo machine architecture
  - C code compiled into BPF code
  - Verified and loaded into kernel
  - Executed under VM runtime
  - Shared BPF maps with userspace

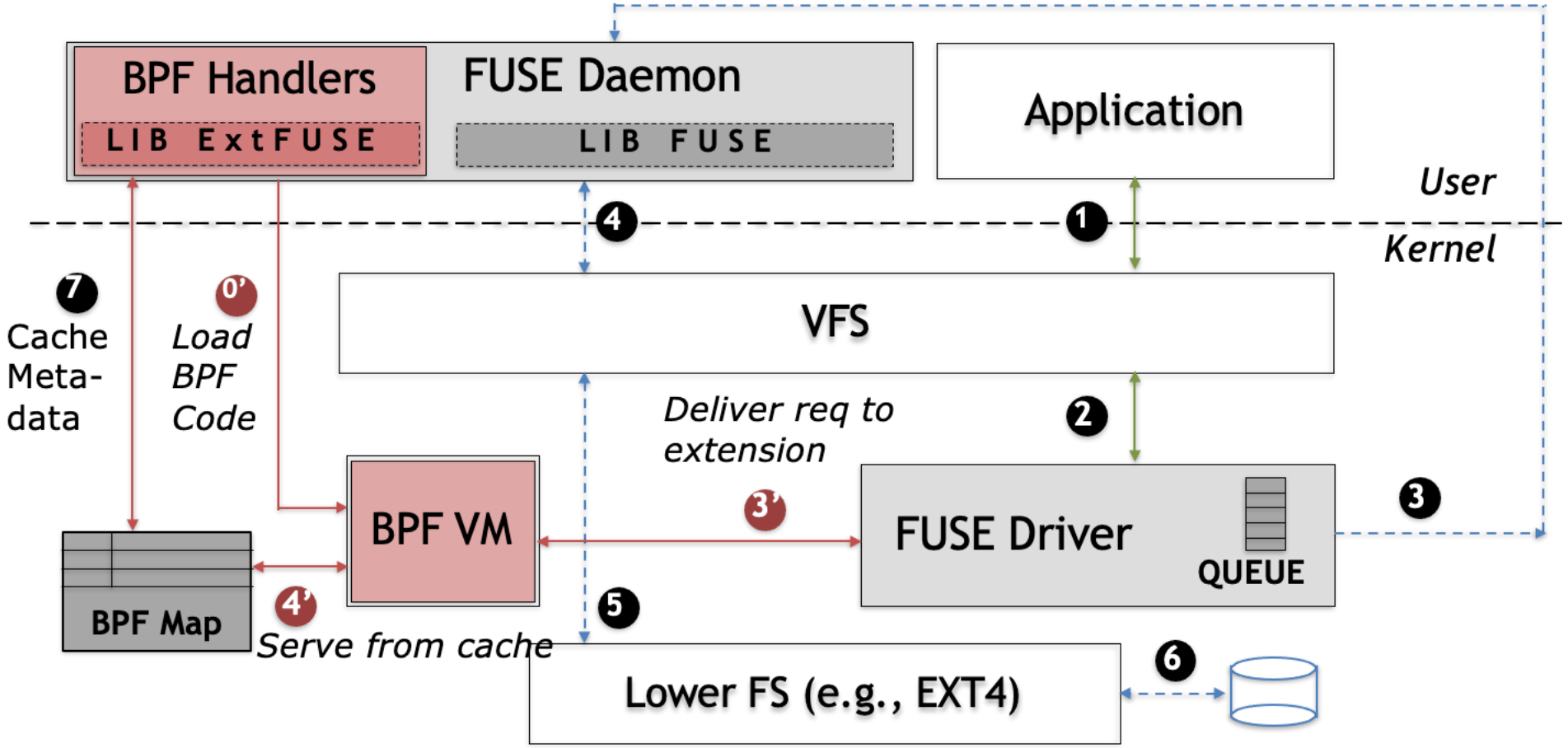


ExtFUSE

- Extension framework for Filesystems in Userspace
  - Register thin extensions
    - Handle requests in kernel
    - Avoid userspace context switch
  - Share data between FUSE daemon and extensions using BPF maps
    - Cache metadata in the kernel

ExtFUSE

- Architecture



ExtFUSE

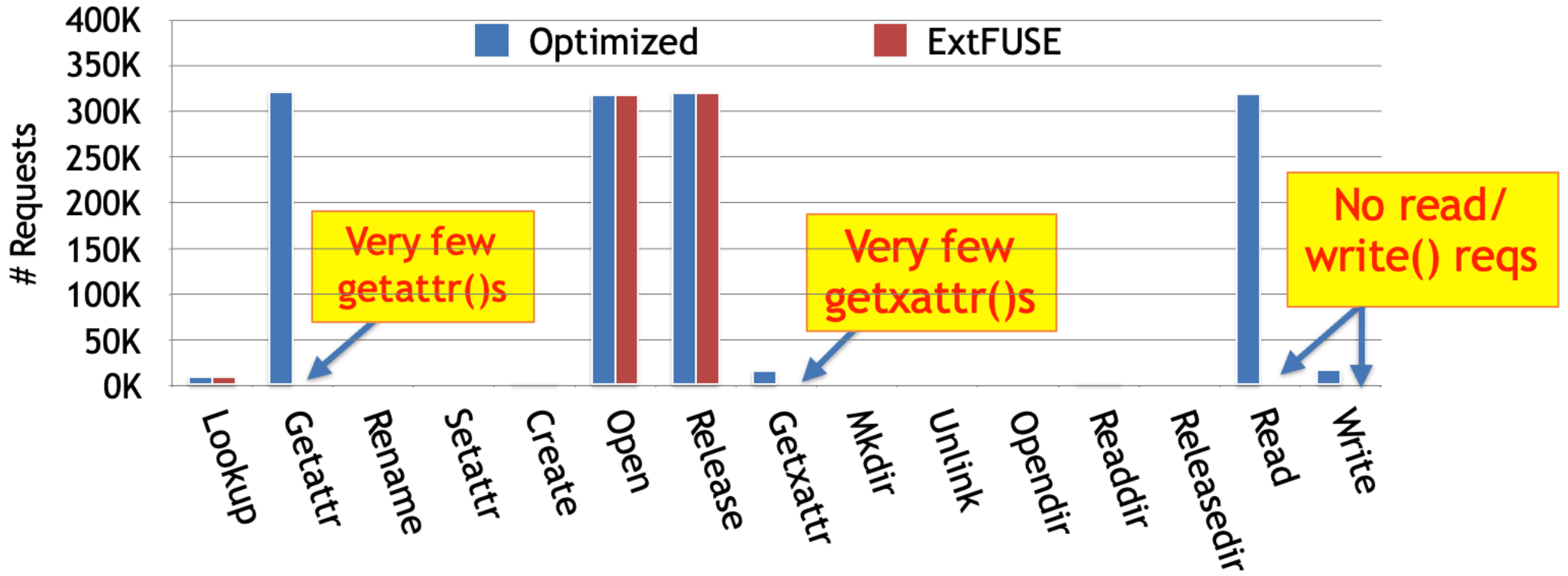
- Examples

```
1  void handle_lookup(fuse_req_t req, fuse_ino_t pino,
2                     const char *name) {
3      /* lookup or create node @cname parent @pino */
4      struct fuse_entry_param e;
5      if (find_or_create_node(req, pino, name, &e)) return;
6  +   lookup_key_t key = {pino, name};
7  +   lookup_val_t val = {0/*not stale*/, &e};
8  +   extfuse_insert_shmap(&key, &val); /* cache this entry */
9      fuse_reply_entry(req, &e);
10 }
```

```
1  int lookup_extension(extfuse_req_t req, fuse_ino_t pino,
2                       const char *name) {
3      /* lookup in map, bail out if not cached or stale */
4      lookup_key_t key = {pino, name};
5      lookup_val_t *val = extfuse_lookup_shmap(&key);
6      if (!val || atomic_read(&val->stale)) return UPCALL;
7      /* EXAMPLE: Android sdcard daemon perm check */
8      if (!check_caller_access(pino, name)) return -EACCES;
9      /* populate output, incr count (used in FUSE_FORGET) */
10     extfuse_reply_entry(req, &val->e);
11     atomic_incr(&val->nlookup, 1);
12     return SUCCESS;
13 }
```

ExtFUSE

- Performance
  - "cd linux-4.18; make tinyconfig; make –j4



XFUSE

* XFUSE: An Infrastructure for Running Filesystem Services in User Space (ATC'21)

Motivation

- Userspace filesystem
  - Benefits
    - Higher development efficiency and velocity
    - Decreased dependency on OS
    - In Cloud Service, providers can offer additional storage services to customers through storage client implemented in userspace
  - Concerns
    - Performance
    - RAS (Reliability, Availability and Serviceability)
    - Application and build changes may be required

Motivation

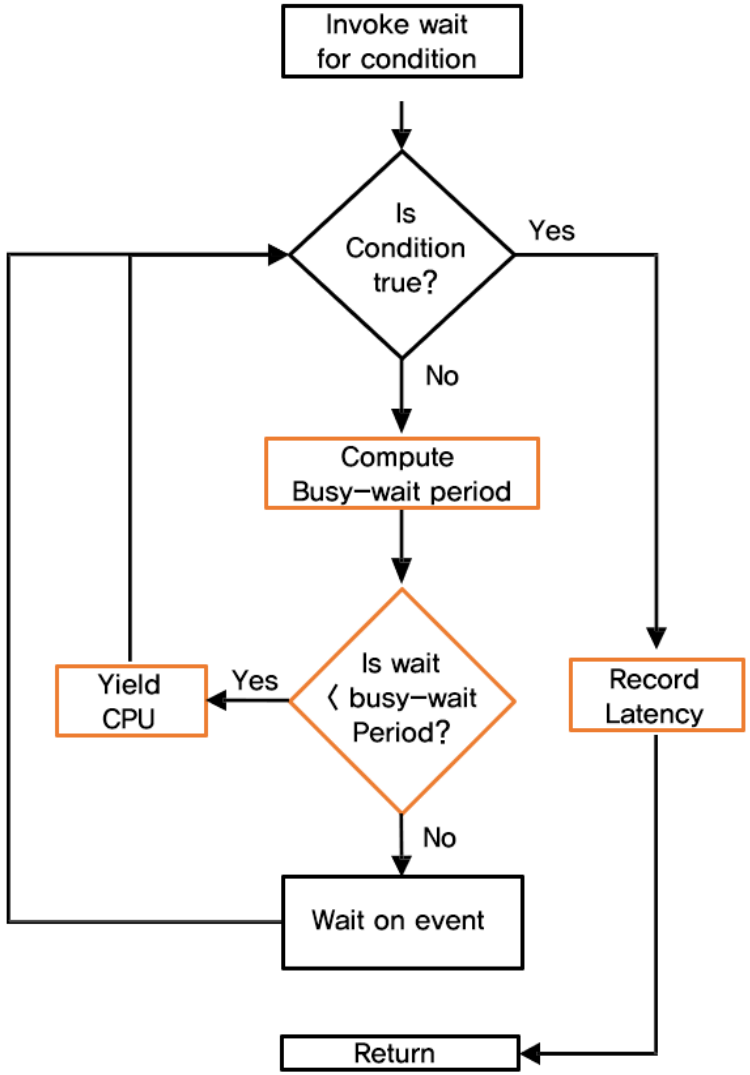
- Backward compatible with FUSE
- Improvement of performance and RAS for XFUSE-optimized filesystems
- Large-scale and gradual rollout in production

- Designed for userspace filesystems that
  - Use high speed storage devices
    - PMEM, fast SSDs, distributed storage systems based on high performance network
  - Are deployed in production environments
    - With strict RAS requirements

XFUSE

- Adaptive waiting
  - Problem
    - Kernel event-wait and notification take a few μs to deliver
    - High perf storages (e.g. PMEM): metadata/data may become available sooner

- Use busy-wait period
  - End-to-end latency can be as low as 3~4 μs with busy waiting (vs. 8~9 μs under event-wait)



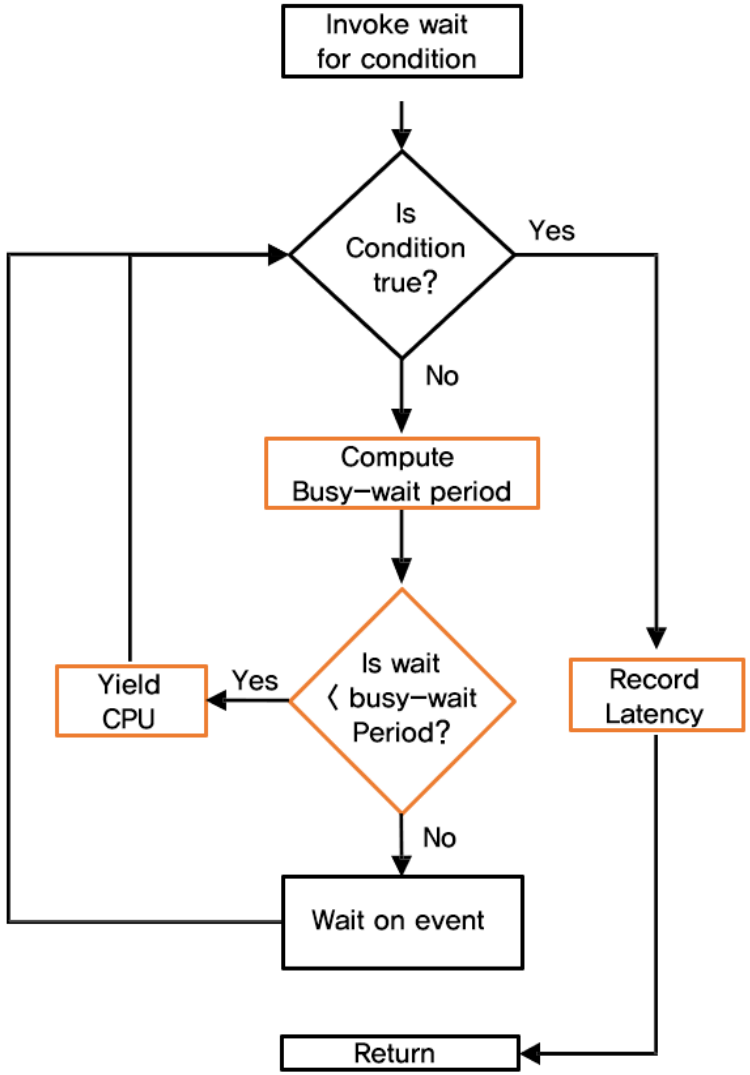
XFUSE

- Adaptive waiting
  - *if (actual time required to service) > (busy-wait period)*
  - Attempting to busy wait is futile and only wastes CPU resources
  - Dynamically predict if busy waiting is beneficial, and
  - Turn on/off busy waiting accordingly

Wait-decision:

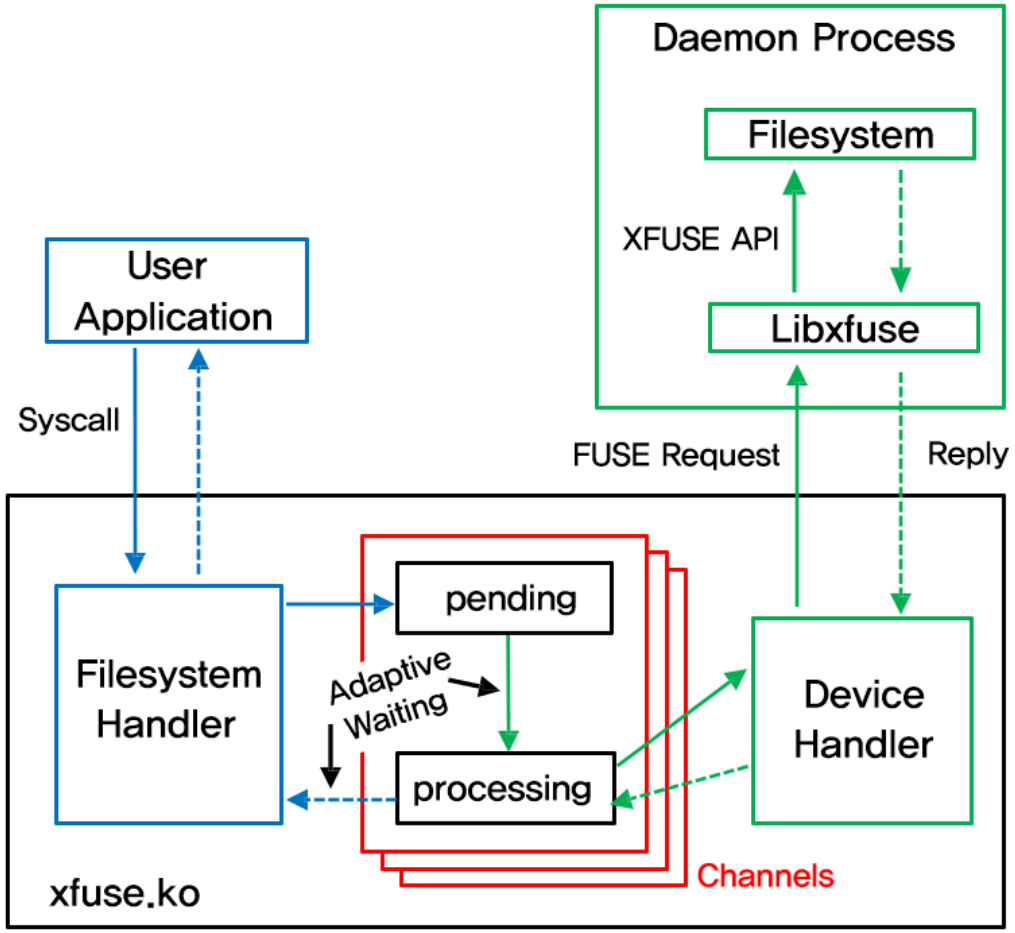
```
threshold = busy_wait_period + event_wait_overhead
          = 10μs + 5μs = 15μs

if observed_latency < threshold
        do busy-event wait
else
        do event wait
```



XFUSE

- Increased parallelism
  - FUSE
    - New request → pending queue (one per mount)
    - Request fetched → processing queue (one per FD)
  - XFUSE
    - Introduces multiple request pending queues
    - Groups each pair of pending and processing queues as a channel
    - New request → channel (per selection policy)



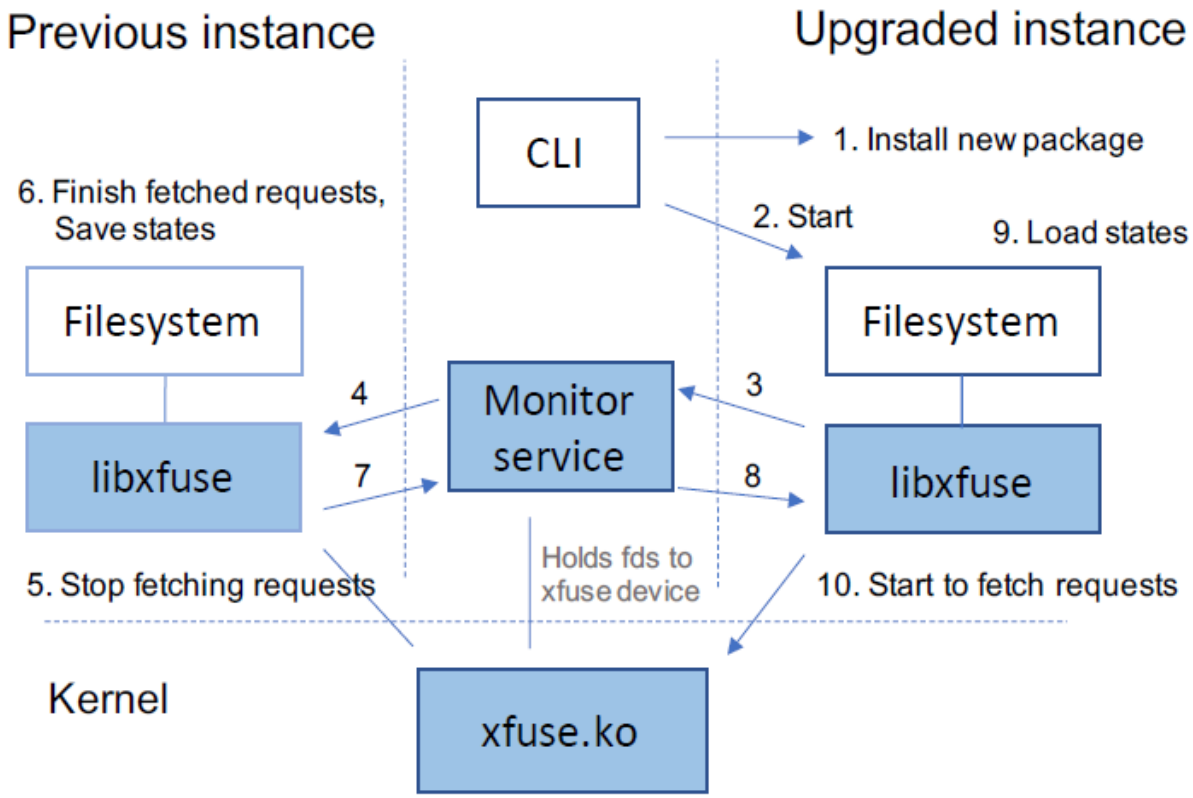
XFUSE

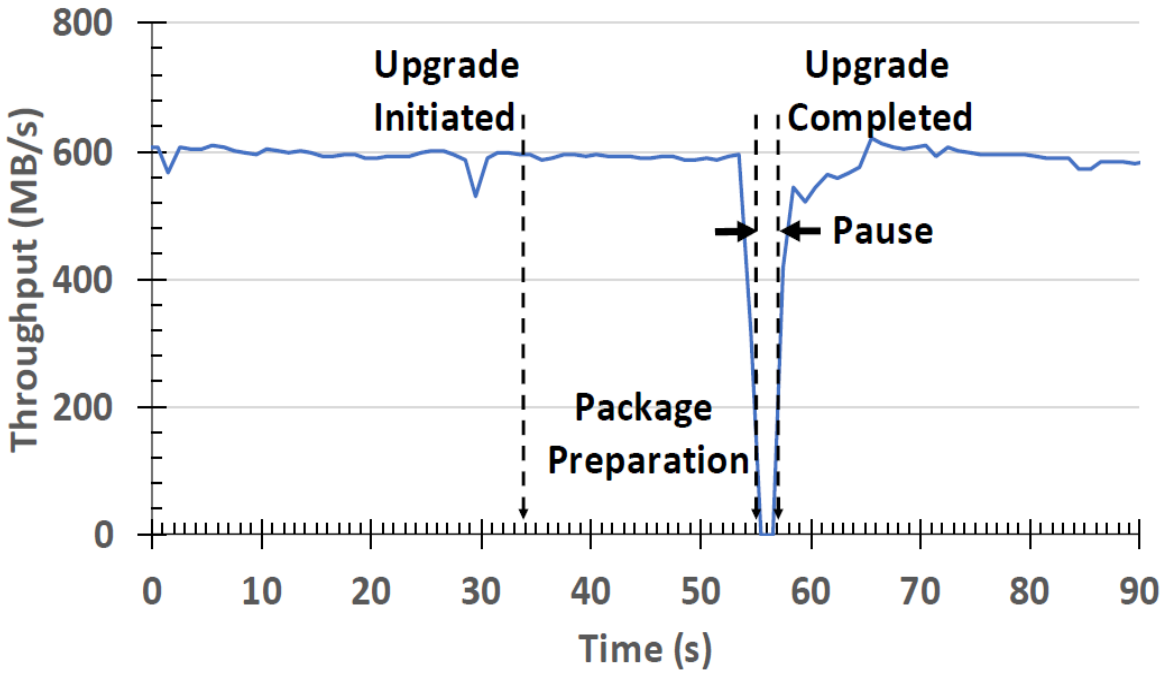
- Business needs
  - Fast paced rollout of new features and bug fixes for userspace filesystems
  - Minimal disruption to tens or hundreds of mounts and apps on each host during upgrade

- Online upgrade
  - Extension to support an online upgrade workflow and a state transition
  - Monitor Service
    - Coordinates the interactions between two filesystem daemons
    - Assists the transfer of filesystem internal states, including FDs (to special fuse device)

XFUSE

- Online upgrade



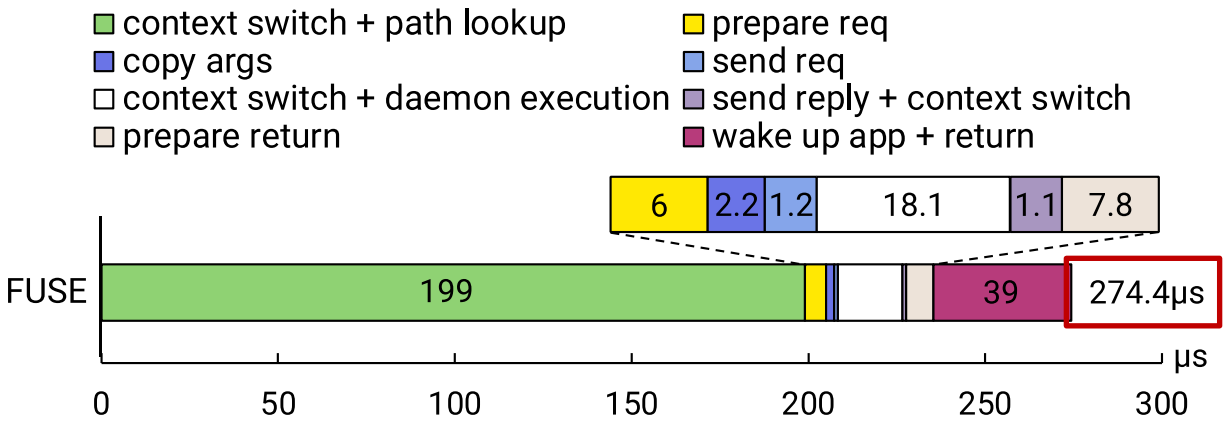


RFUSE

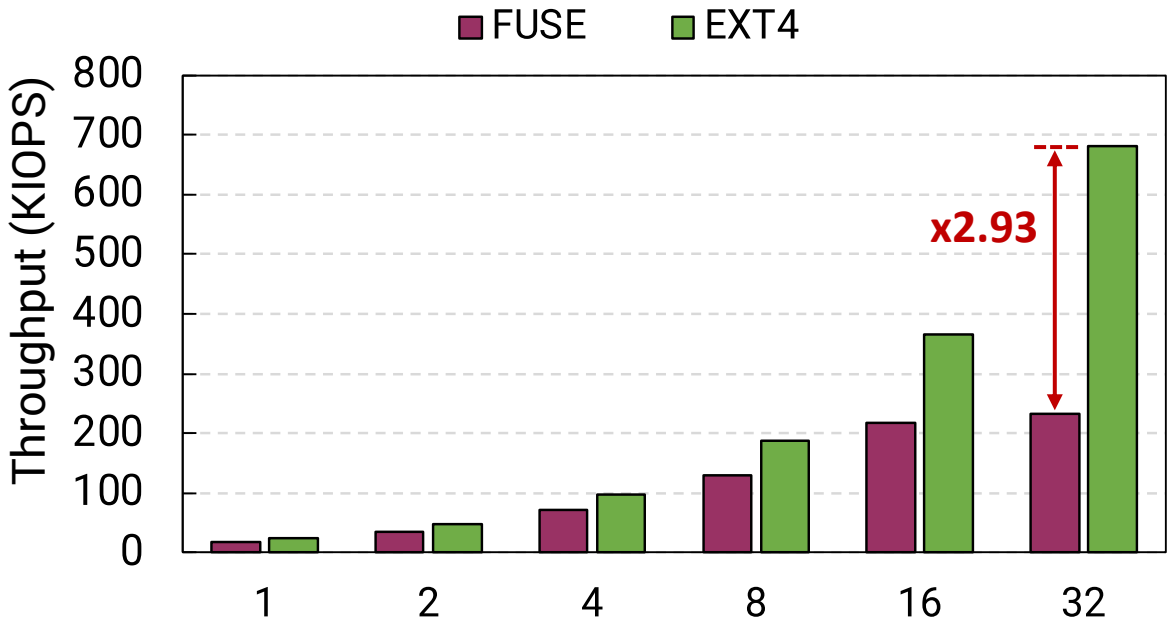
* RFUSE: Modernizing Userspace Filesystem Framework through Scalable Kernel-Userspace Communication (FAST'24)

Motivation

- FUSE overhead
  1. Long latency of no-op request handling
  2. Low scalability of random read on StackFS



<Latency overhead>



<Scalability Issue>

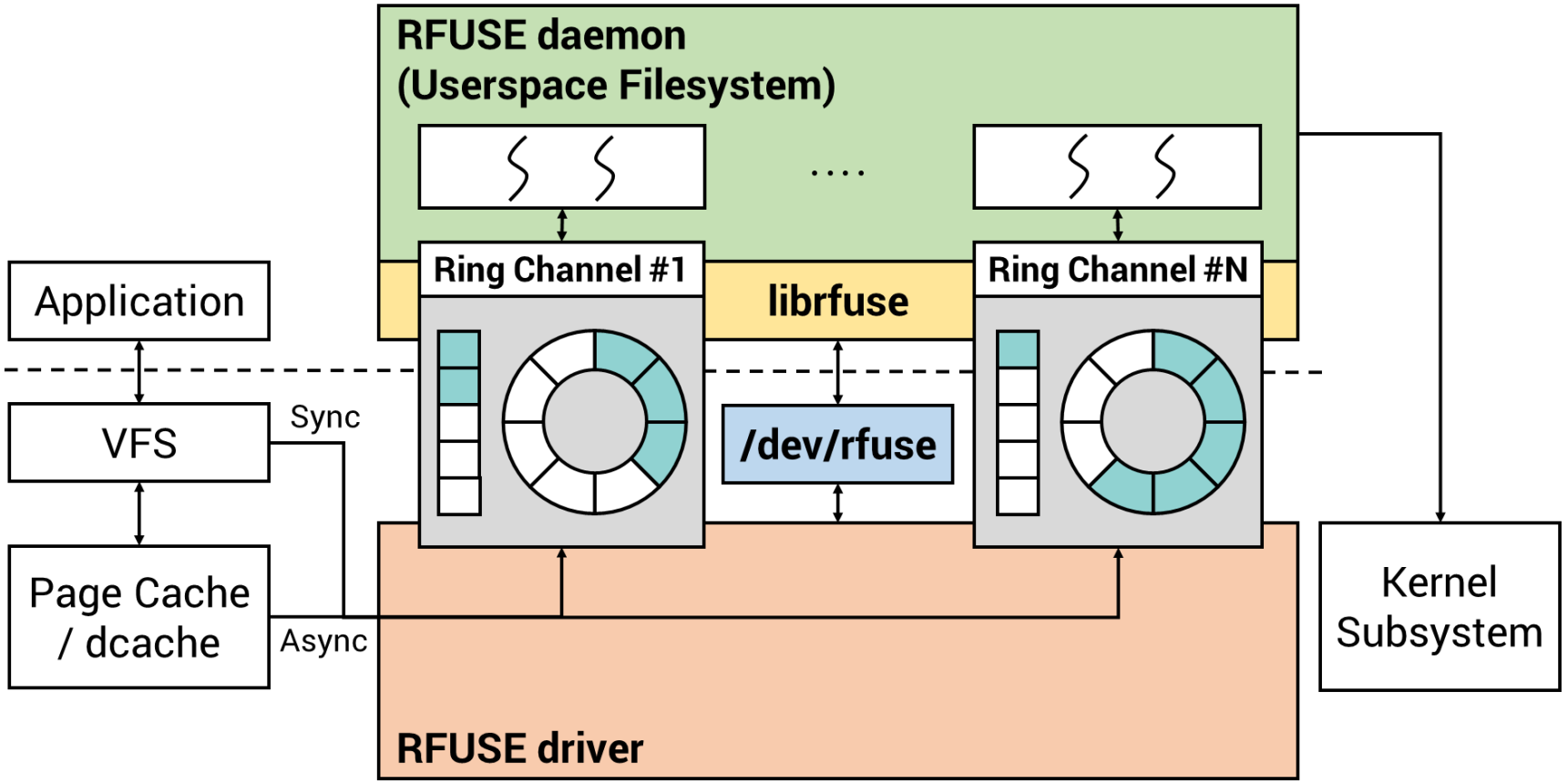
Motivation

- The SOTA studies focus on enhancing communication between the kernel and userspace, aiming for performance on par with in-kernel filesystems

- However, they are only partially effective because:
  - They often require additional development efforts, which demonstrate low compatibility with existing FUSE-based filesystems
  - They still relies on a copy-based communication on single pending queue

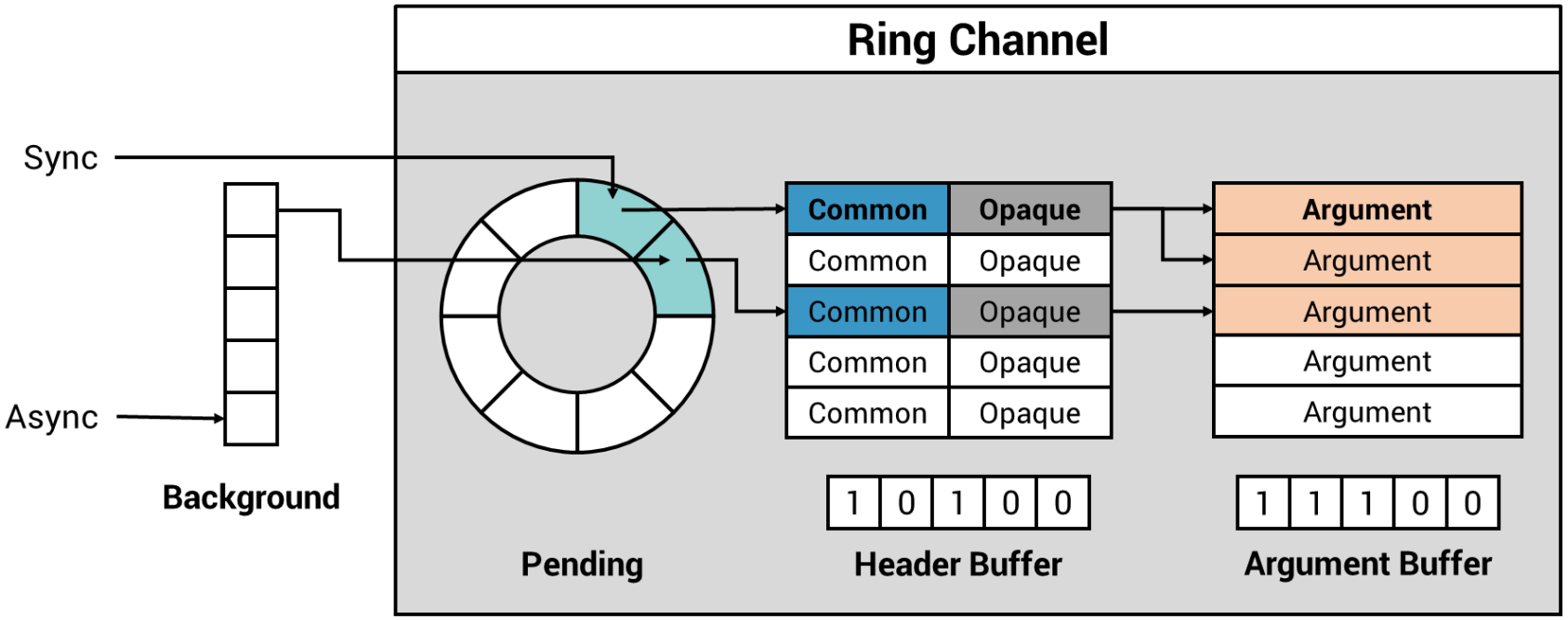
RFUSE

- Architecture



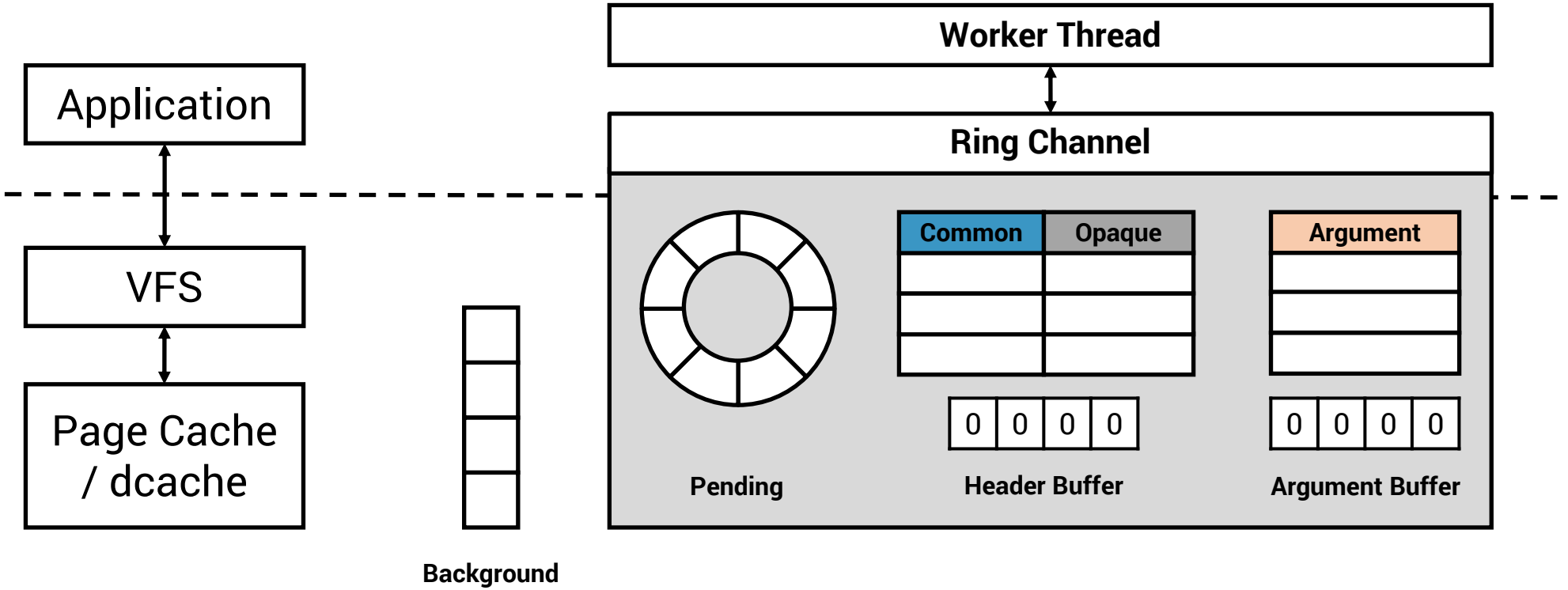
RFUSE

- Per-core ring channel



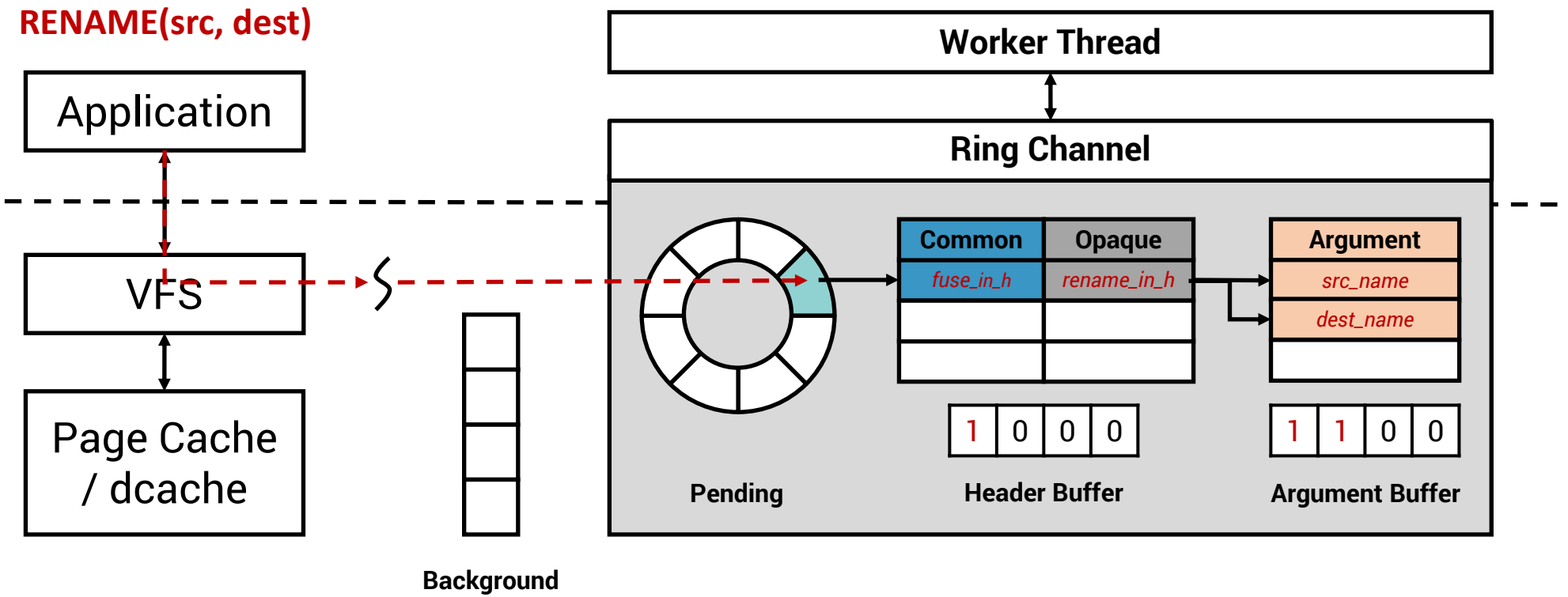
RFUSE

- Per-core ring channel



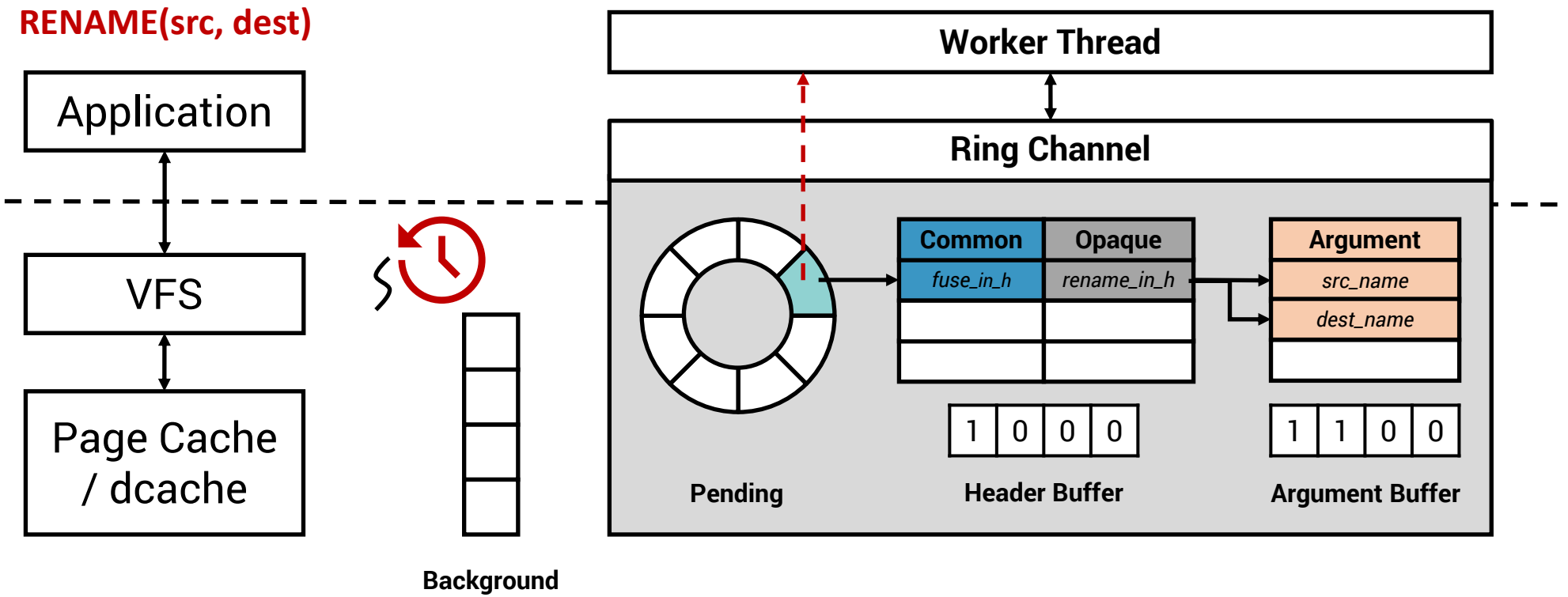
RFUSE

- Per-core ring channel



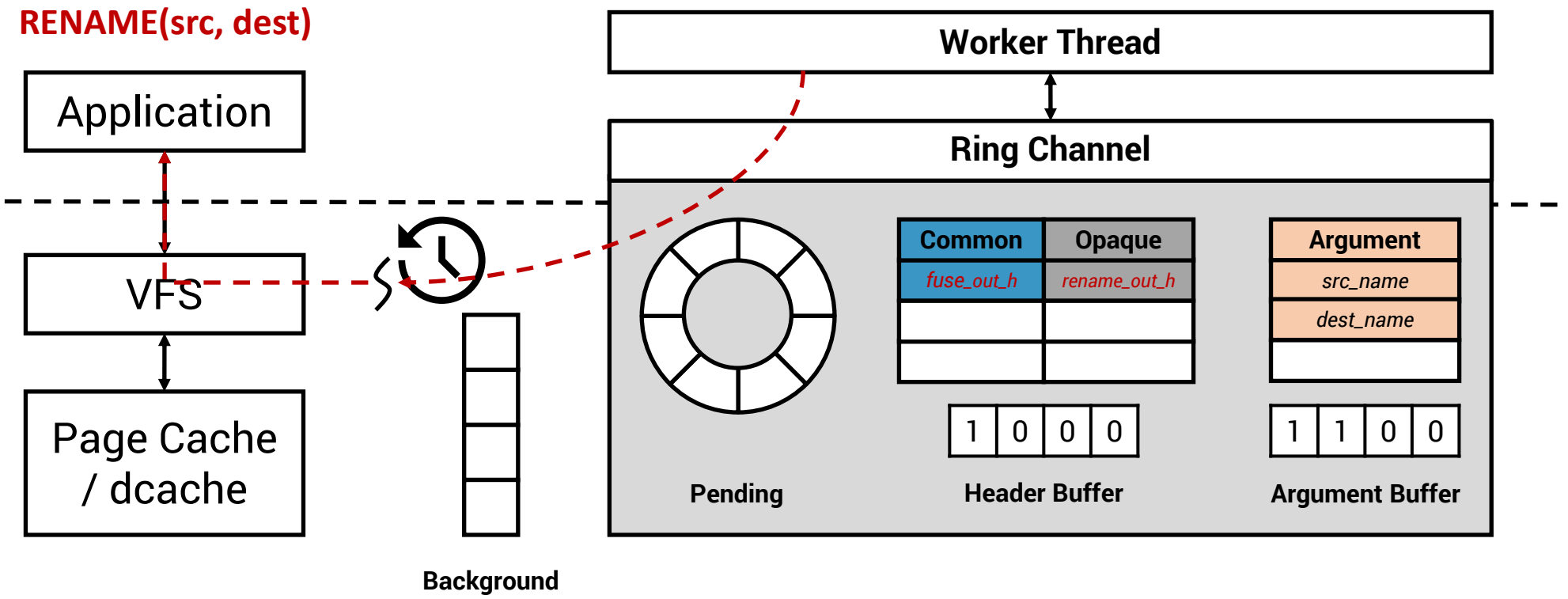
RFUSE

- Per-core ring channel



RFUSE

- Per-core ring channel



RFUSE

- Full compatibility with FUSE
  - No modifications of all FUSE APIs exposed to developers
    - Both high-level FUSE API and low-level FUSE API
    - Splicing I/O interface

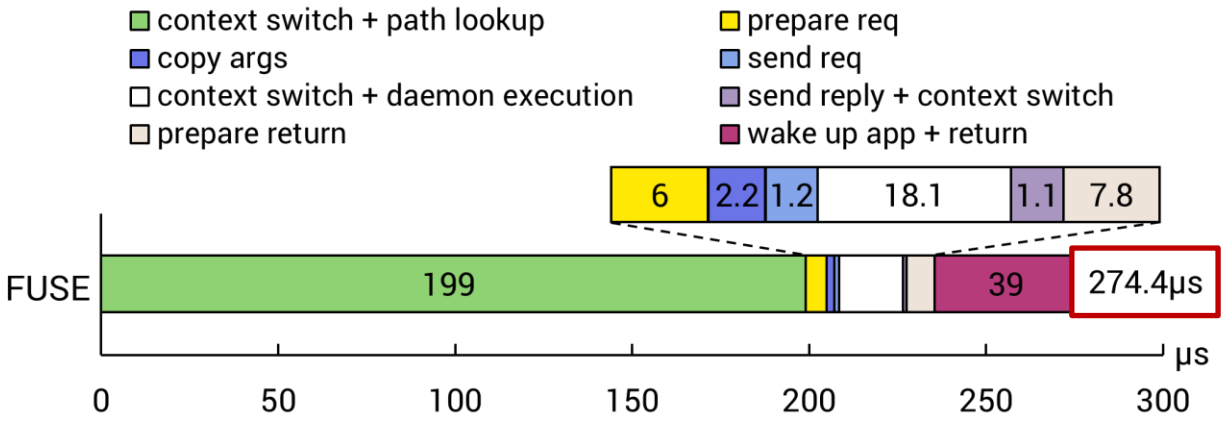
```
struct fuse_operations {
        .getattr    = …
        .readlink   = …
        .mkdir      = …
        ... }
```

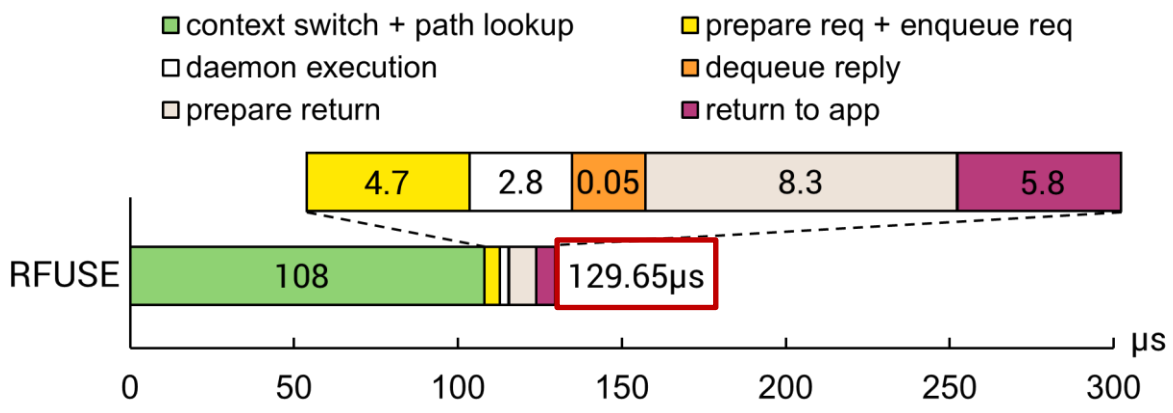
```
struct fuse_lowlevel_ops {
        .init       = …
        .destroy    = …
        .lookup     = …
        ... }
```

  - Users do not need to rewrite their FUSE-based filesystem code when using RFUSE.

RFUSE

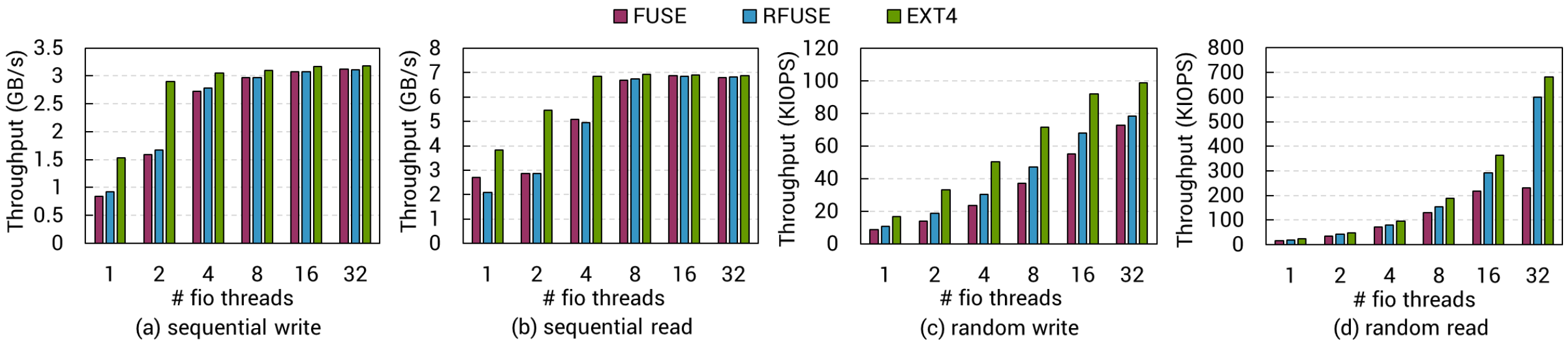
- Latency overhead
  - CREAT() on root directory, which promptly returns without performing any action





RFUSE

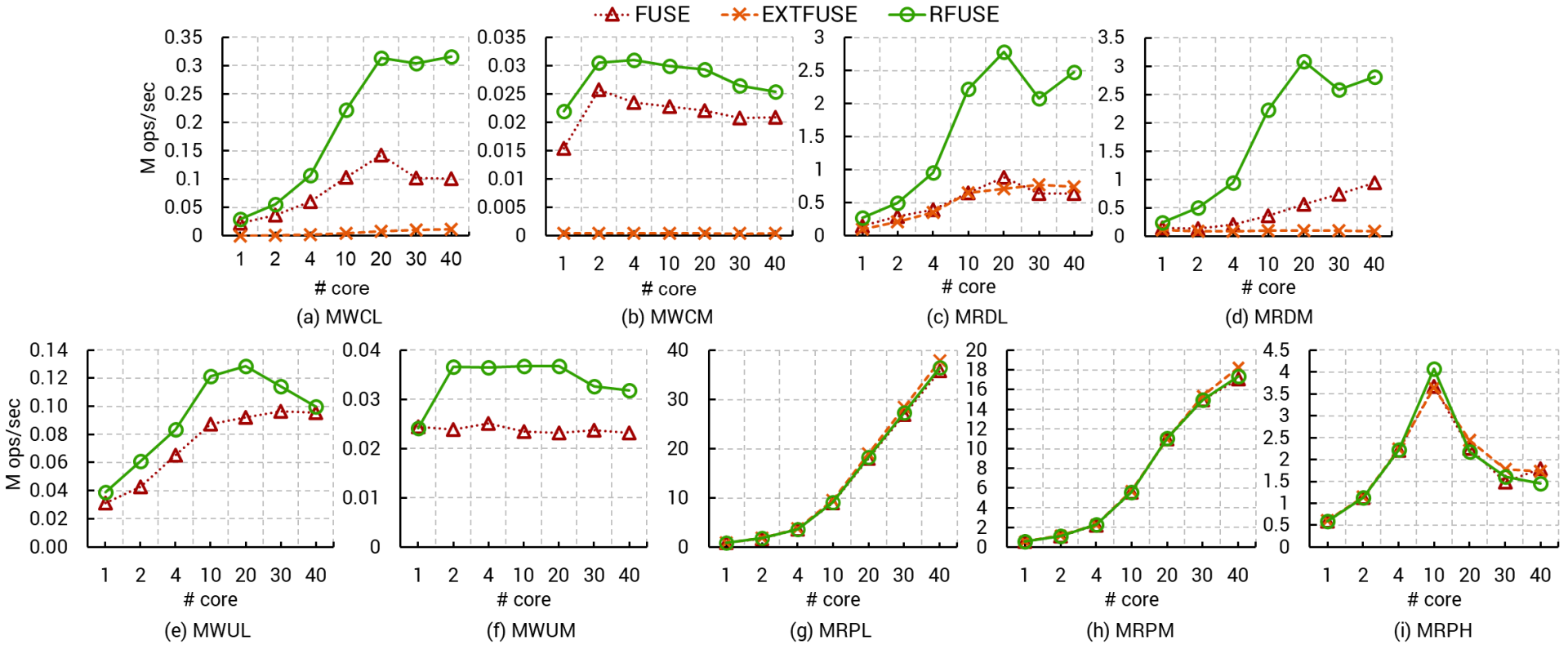
- I/O performance
  - FIO benchmark on StackFS while increasing the number of threads
    - Sequential I/O with 128KB size
    - Random I/O with 4KB size
    - 128GB file size in total



RFUSE

- Metadata performance
  - FXMARK benchmark on StackFS

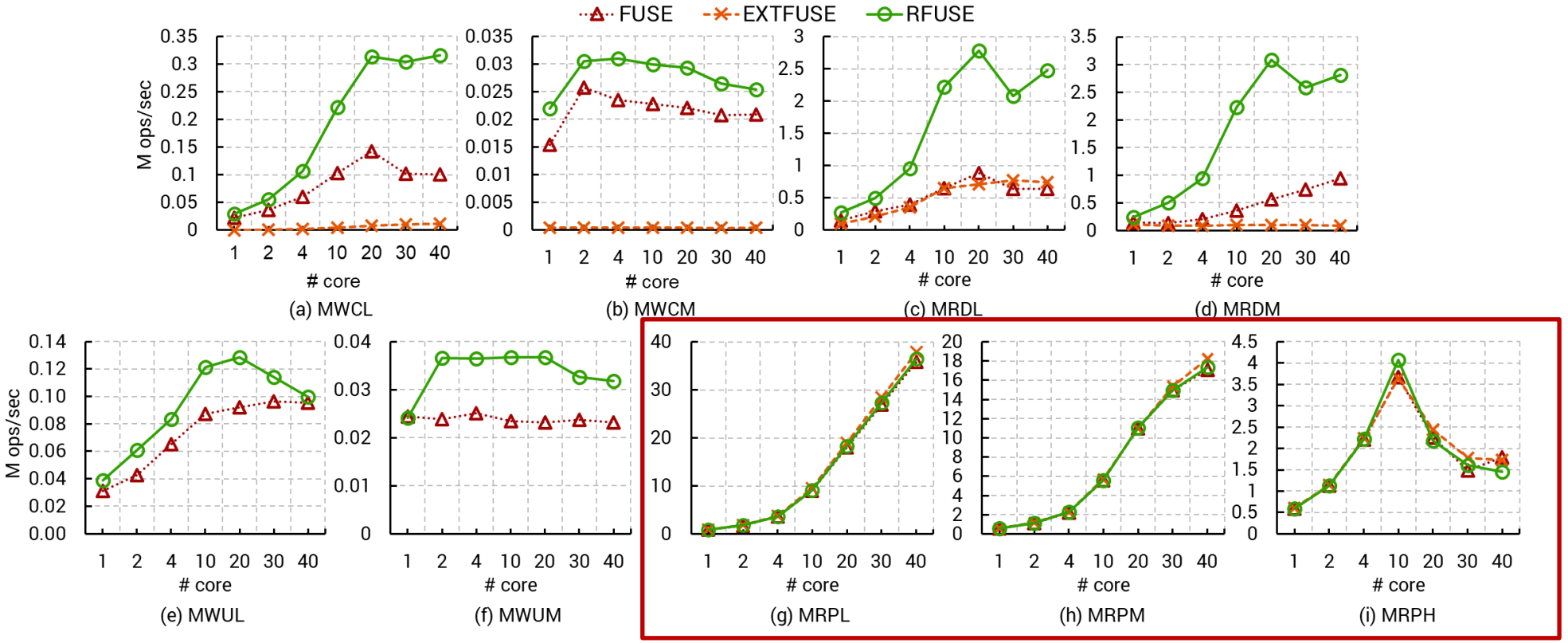
Workload	Description
MWCL	Create empty files in a private directory
MWCM	Create empty files in a shared directory
MRDL	Enumerate a private directory
MRDM	Enumerate a shared directory
MWUL	Unlink empty files in a private directory
MWUM	Unlink empty files in a shared directory
MRPL	Open and close private files in a directory
MRPM	Open and close arbitrary files in a directory
MRPH	Open and close the same file in a directory



RFUSE

- Metadata performance
  - FXMARK benchmark on StackFS

Workload	Description
MWCL	Create empty files in a private directory
MWCM	Create empty files in a shared directory
MRDL	Enumerate a private directory
MRDM	Enumerate a shared directory
MWUL	Unlink empty files in a private directory
MWUM	Unlink empty files in a shared directory
MRPL	Open and close private files in a directory
MRPM	Open and close arbitrary files in a directory
MRPH	Open and close the same file in a directory



affected by dcache

Thank you