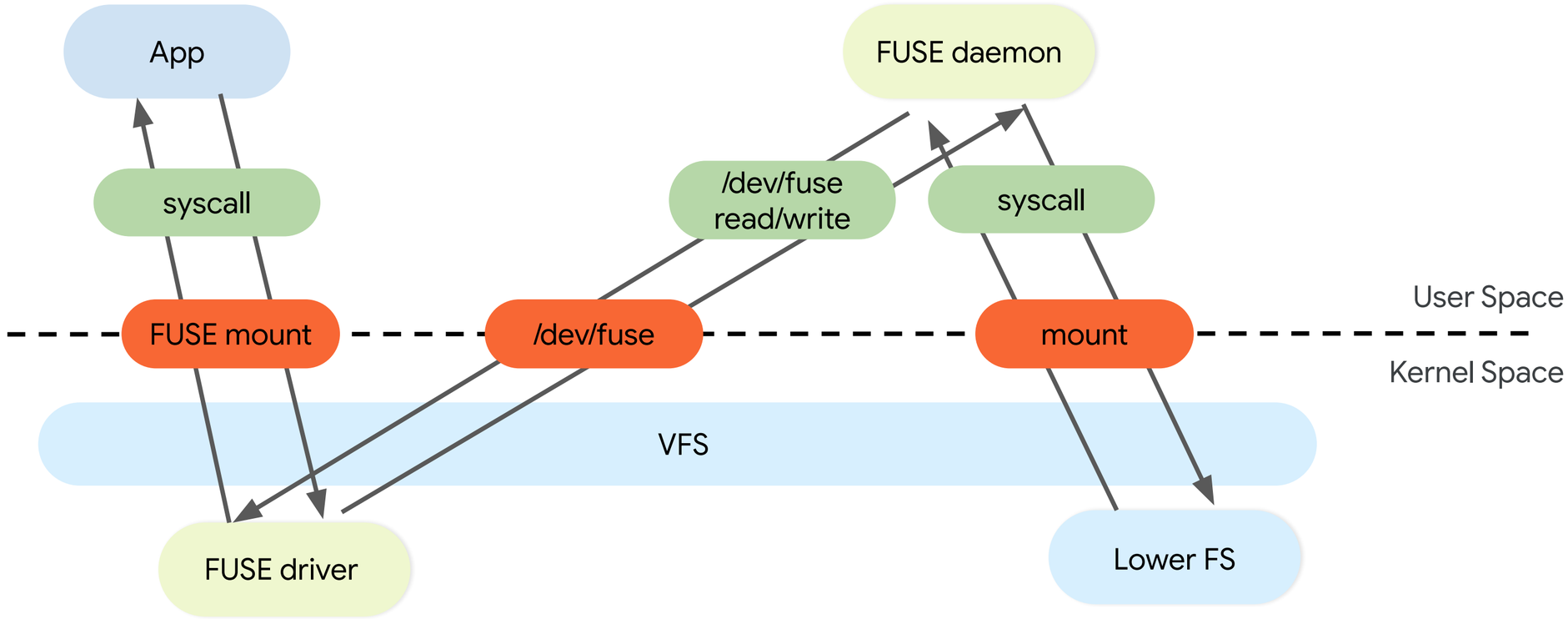
Fuse-BPF

Stacked Filesystem Support for FUSE

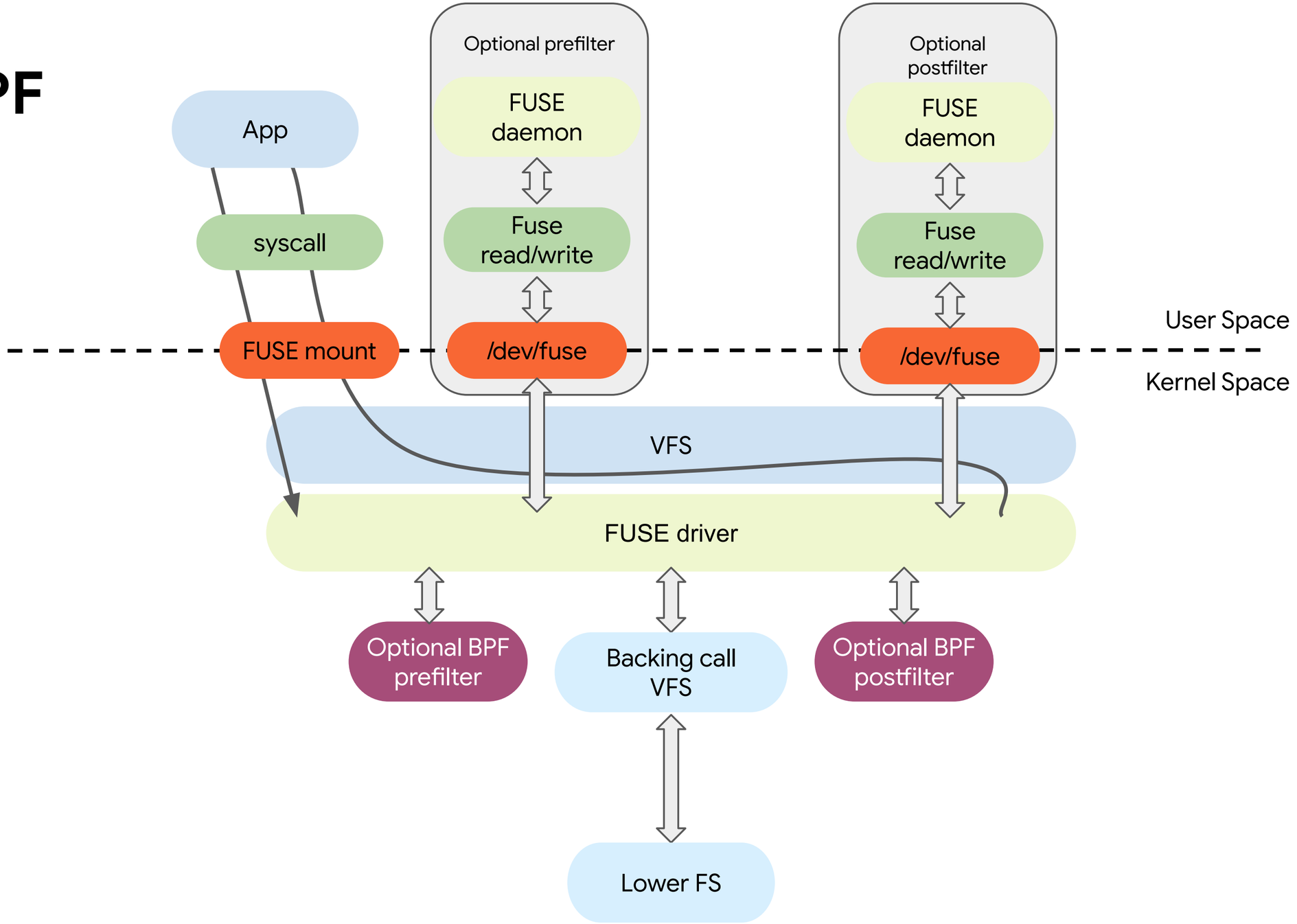
Design Goals

- Optimize FUSE when used as a Stacked Filesystem
  - Perform as close to native filesystem as possible
- Keep ease of use of FUSE
  - Should maintain compatibility across kernel versions

Classic Fuse



Fuse BPF


Optional prefilter
FUSE daemon
Fuse read/write
/dev/fuse
Optional postfilter
FUSE daemon
Fuse read/write
/dev/fuse
App
syscall
FUSE mount
User Space
Kernel Space
VFS
FUSE driver
Optional BPF prefilter
Backing call VFS
Optional BPF postfilter
Lower FS

Current Design

- BPF struct op program set at lookup time
  - Two callbacks per opcode: prefilter and postfilter
- Prefilter
  - Adjust input arguments (structs when possible, fuse_buffer when variable sized)
  - Can call userspace prefilter, or abort and take normal fuse path
- Postfilter
  - Adjust output arguments
  - Can call userspace postfilter and adjust error code
- Can implement bpf callbacks as needed
  - If no bpf is present, will default to unaltered backing call (In future this may be configured in fuse_init for forwards compatibility)

Fuse buffers

- Opaque structure
  - Can only be altered by kfunc
  - Must convert to dynptr using kfuncs before use (bpf_fuse_get_rw_dynptr/bpf_fuse_get_ro_dynptr) name will likely change
  - Contain metadata describing argument’s possible size and allocation status. Fuse cleans up extra allocations after calls.

Using Fuse BPF

- Register bpf program
  - bpftool struct_op register [compiled bpf]
- Set BPF at mount time
  - root_bpf=[name of fuse_ops]
  - root_dir=[fd for backing directory]
- Set at lookup time
  - Append block to lookup response

uint32_t entry_type	FUSE_ENTRY_BPF
uint32_t reserved	0
char name[16]	“struct_opname”
uint32_t entry_type	FUSE_ENTRY_BACKING
uint32_t reserved	0
uint64_t unused	0
Uint64_t fd	[fd of backing file]

Fuse_ops structure

```
struct fuse_ops {

    uint32_t (*default_filter)(const struct bpf_fuse_meta_info *meta);

    uint32_t (*open_prefilter)(const struct bpf_fuse_meta_info *meta, struct fuse_open_in *in);
    uint32_t (*open_postfilter)(const struct bpf_fuse_meta_info *meta, const struct fuse_open_in *in,
                                                                    struct fuse_open_out *out);

    uint32_t (*opendir_prefilter)(const struct bpf_fuse_meta_info *meta, struct fuse_open_in *in);
    uint32_t (*opendir_postfilter)(const struct bpf_fuse_meta_info *meta, const struct fuse_open_in *in,
                                                                    struct fuse_open_out *out);

    …

    char name[BPF_FUSE_NAME_MAX];

};
```

Performance

Hardware

- ASRock Industrial AMD Barebone 4X4 BOX-4300U
- Kingston FURY Impact 16GB (1 x 16GB) 260-Pin DDR4 SO-DIMM DDR4 3200 (PC4 25600)
- Western Digital WD_BLACK SN770 M.2 2280 250GB PCIe Gen4 16GT/s

Software

- Ubuntu 22.10

Lower filesystem is a tmpfs ramdisk

All reported values in MiB/s

Bpf prog (v1) Simple BPF	Fuse lower	LibFuse passthrough_hp		Fuse BPF lower	Fuse BPF	
fio-seq-read	3,514.33	1,571.00	-55.30%	3,409.00	3,433.67	0.72%
fio-rand-RW: READ	3,116.67	246.33	-92.10%	3,102.00	2,888.33	-6.89%
fio-rand-RW: WRITE	2,078.33	164.33	-92.09%	2,068.67	1,926.33	-6.88%
filecreate-ioengine	16.37	13.53	-17.31%	16.33	15.93	-2.45%

Bpf prog (v1) Tracing BPF	Fuse lower	LibFuse passthrough_hp		Fuse BPF lower	Fuse BPF	
fio-seq-read	3,410.00	1,584.33	-53.54%	3,430.67	3,413.00	-0.51%
fio-rand-RW: READ	3,094.00	214.00	-93.08%	3,078.00	2,412.33	-21.63%
fio-rand-RW: WRITE	2,063.33	142.67	-93.09%	2,052.67	1,608.33	-21.65%
filecreate-ioengine	16.17	14.07	-12.99%	17.20	15.97	-7.17%

Struct_op version	Fuse lower	LibFuse passthrough_hp		Fuse BPF lower	Fuse BPF	
fio-seq-read	3,468.00	1,589.00	-54.18%	3,503.00	3,454.00	-1.40%
fio-rand-RW: READ	3,132.67	246.33	-92.14%	3,129.33	2,582.67	-17.47%
fio-rand-RW: WRITE	2,089.00	164.00	-92.15%	2,086.67	1,722.00	-17.48%
filecreate-ioengine	16.27	13.73	-15.57%	16.10	15.70	-2.48%

ToDo

- Perform operations using fuse daemon credentials
- Support for additional opcodes (FUSE_IOCTL, mapping, etc)
- Verify and use altered arguments from all pre/postfilters
- Support multiple backing files, and maybe multiple bpf programs
- Invalidate dynptrs when fuse_buffer reallocats memory (New dynptr type?)
- FUSE_INIT information for supported backing ops/default actions
- Libfuse integration

Issues

- Does not gracefully fall back to normal fuse
  - Currently bpf created nodes have node id of 0, unless assigned in bpf.
  - Add ability to reserve id block? Provide means to identify node/set node id later?
- Struct ops has no module support
  - Current patch set exports registration call
  - Probably will add module support to struct_ops in the future
- Many struct op callbacks
  - Consolidate some opcodes?
  - Merge pre/post filter and add backing calls as kfuncs?
- Limited dynptr API
  - Many recent updates, patches posted

Upstreaming plans

- Support subset of ops independently
  - Must inform userspace of what ops are supported (FUSE_INIT extension?)
- Bpf changes submitted separately
  - Dynpointer adjustments
  - Struct op module support
    - Fuse struct_op definition will then move to fs/fuse