Balancing Dirty Pages
with I/O throttle

Practical Class 11

Systems and Storage Laboratory
Department of Computer Science and Engineering
Chung-Ang University

What is Page?

- **Page (virtual page) is a fixed-length contiguous block (mostly 4KB) of virtual memory**
  - Smallest unit of data for memory management in a virtual memory operating system

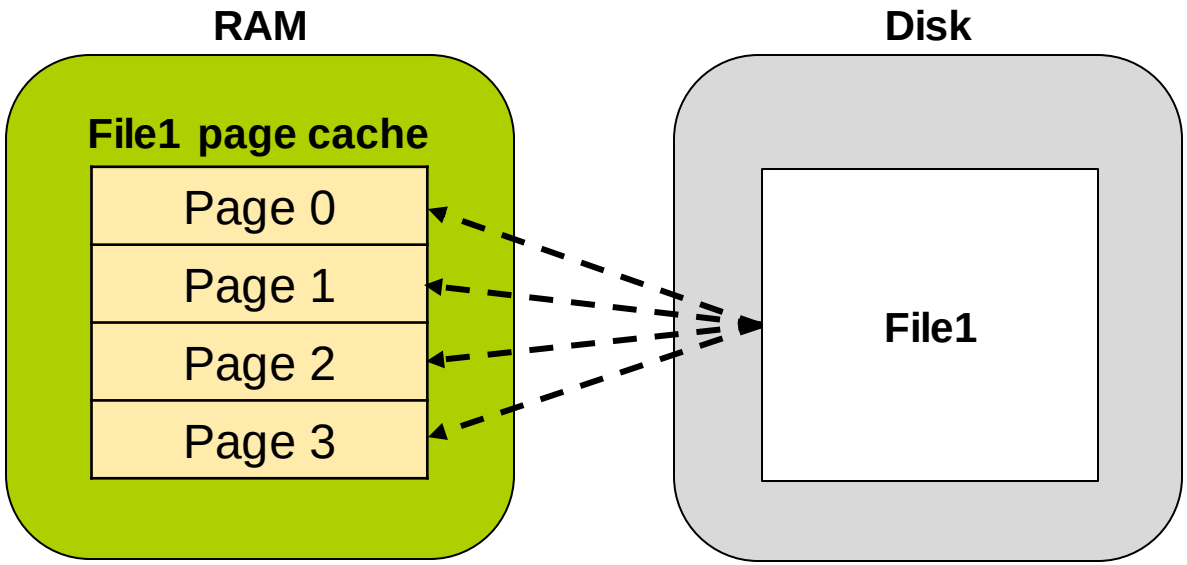


Fig 1. File-mapped pages

Load File from Disk

- **The physical memory is volatile**
  - The common case for getting data into the memory is to read it from files

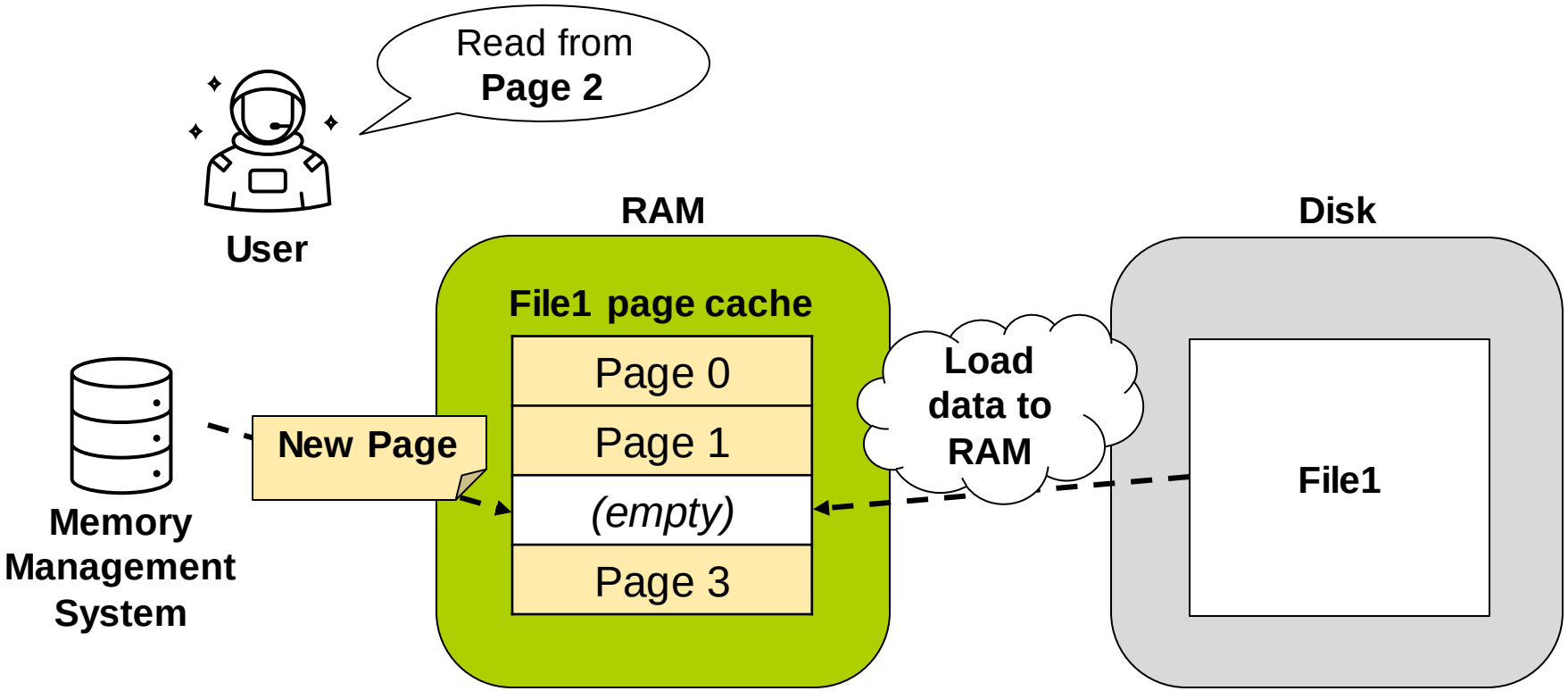


Fig 2. Read file data for the first time

Page Cache

- **Whenever a file is read, the data is put into the page cache**
  - To avoid expensive disk access on the subsequent reads

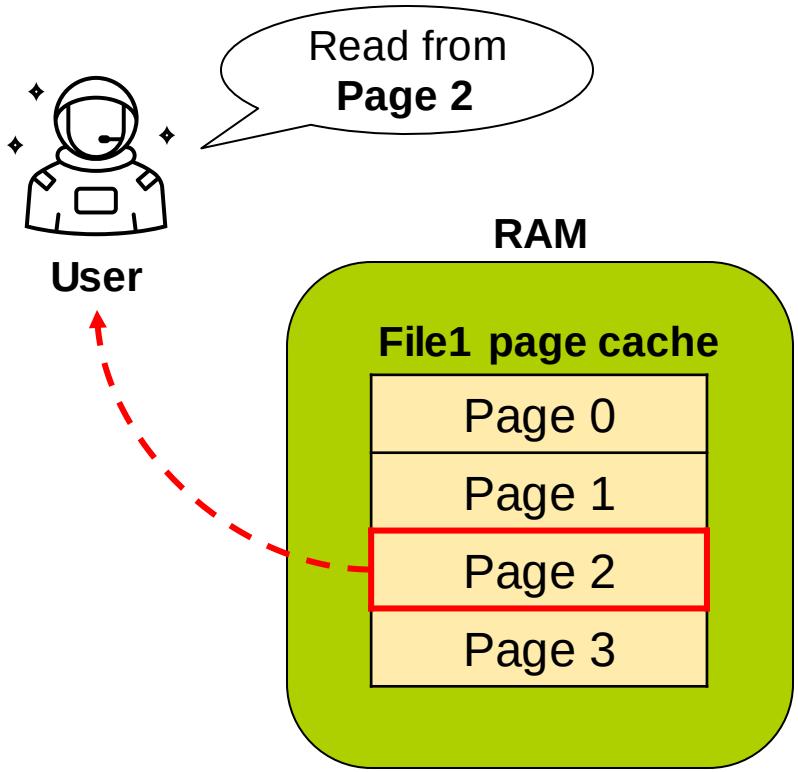


Fig 3. Read file data already on RAM

Page Cache

- **Similarly, when one writes to a file,**
  - The data is placed in the page cache and eventually gets into the backing storage device

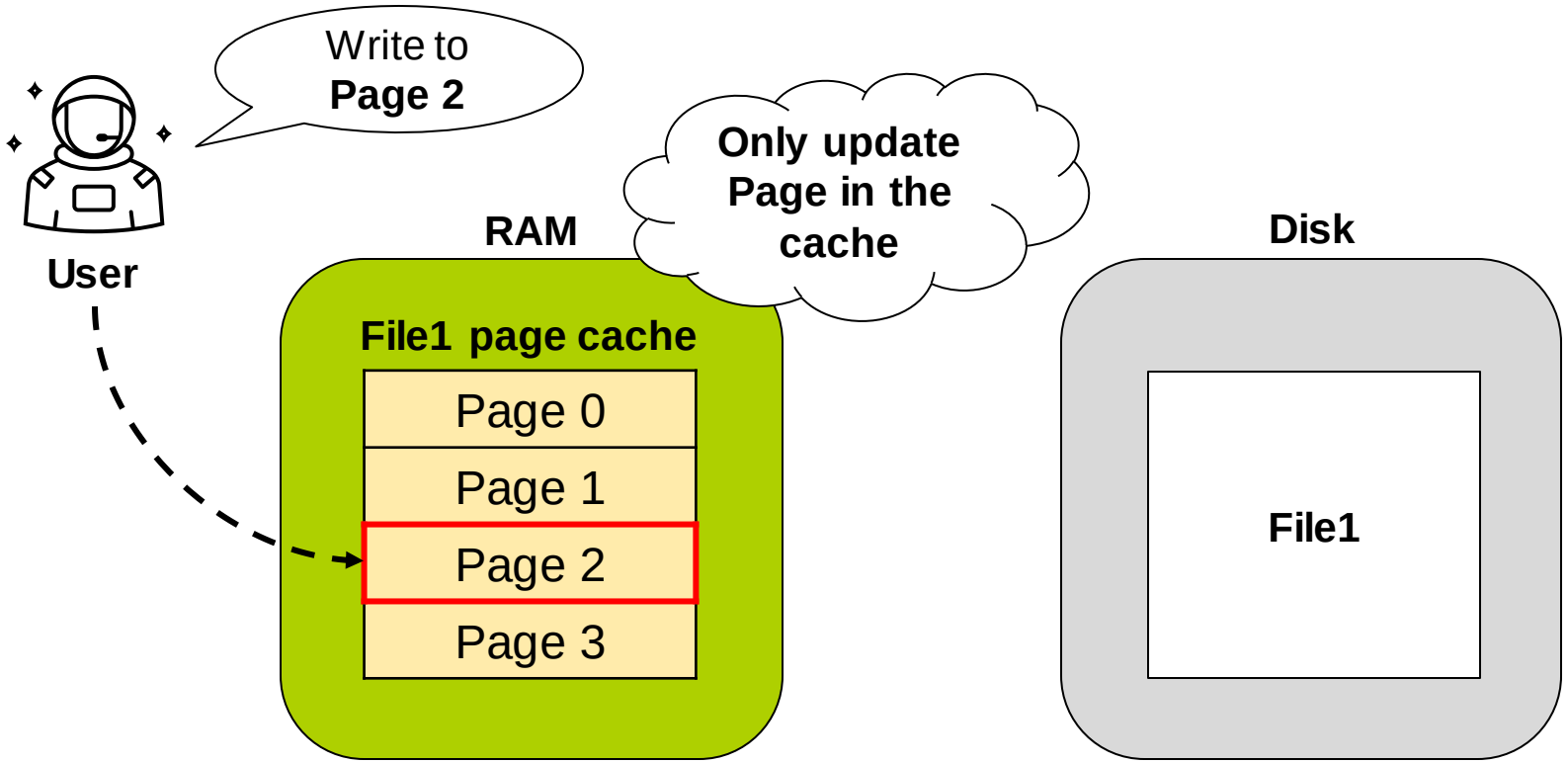


Fig 4. Write file data already on RAM

Writeback

- **The written pages are marked as dirty**
  - When Linux decides to reuse them for other purposes, it makes sure to synchronize the file contents on the device with the updated data.

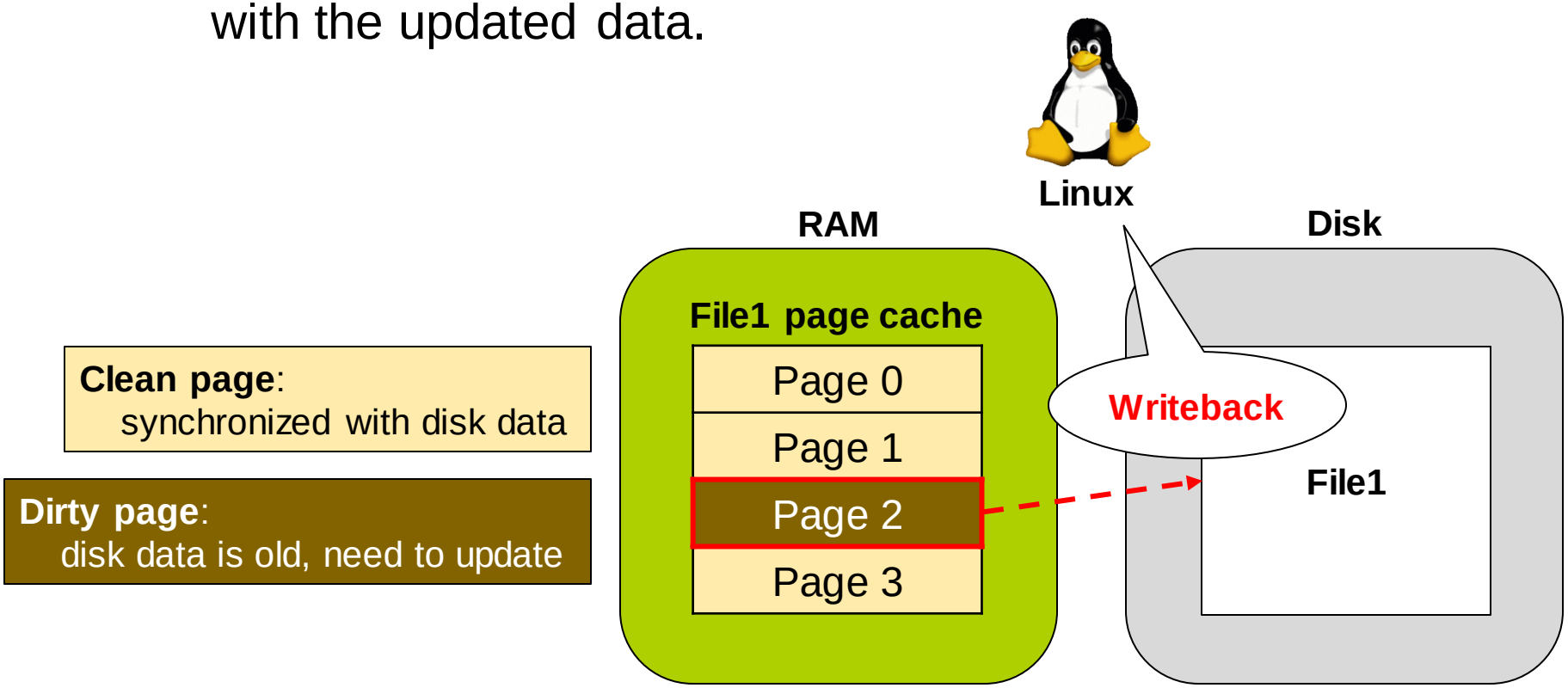


Fig 5. Dirty page and writeback

User space write

Open system call

- Opens the file specified by pathname

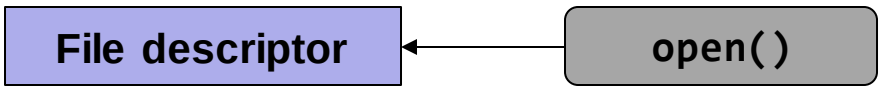
```
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags,
umode_t, mode)
{
    if (force_o_largefile())
        flags |= O_LARGEFILE;

    return do_sys_open(AT_FDCWD, filename, flags, mode);
}
```

  - Return value: a file descriptor

File descriptor

- A small, non-negative integer that is an index to an entry in the process's table of open file descriptors



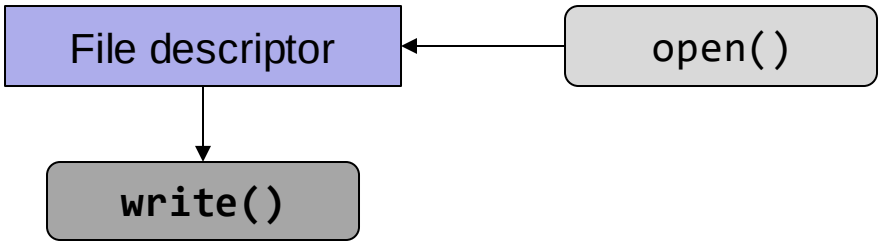
User space write

❖ Write system call

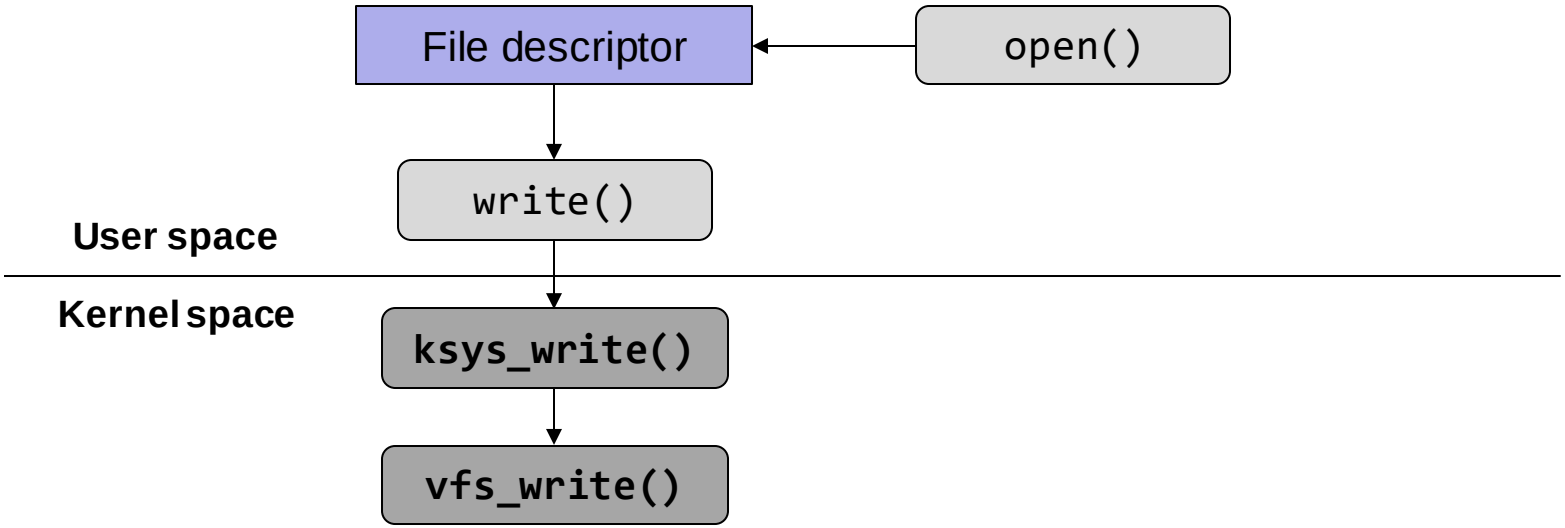
- Writes up to *count* bytes from the buffer starting at *buf* to the file referred to by the file descriptor *fd*

```
SYSCALL_DEFINE3(write, unsigned int, fd, const char __user *, buf,
        size_t, count)
{
    return ksys_write(fd, buf, count);
}
```

  - Return value
    - ✓ On success: the number of bytes written is returned
    - ✓ On error: -1 is returned, and errno is set to indicate the error

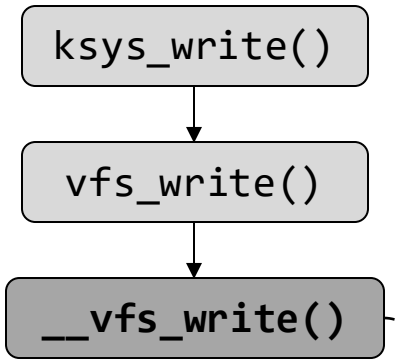


Kernel space write



- **ksys_write()**
  - Get file pointer position and pass it to `vfs_write()`
  - Update file pointer position after `vfs_write()` performs
- **vfs_write()**
  - Check validity of file write
  - Mark file as “currently writing” with `file_start_write()` and `file_end_write()`

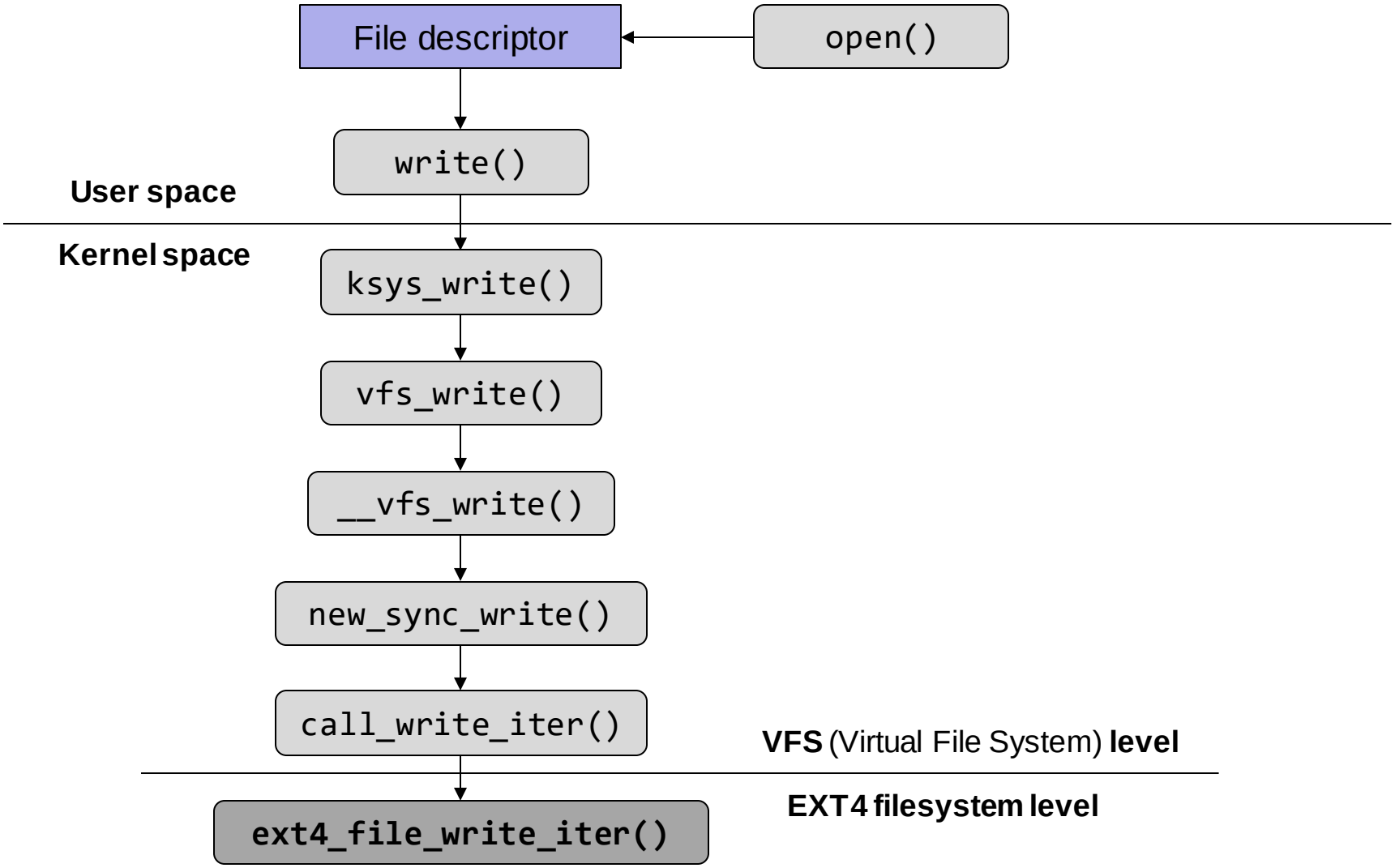
Kernel space write




```
static ssize_t __vfs_write(struct file *file, const char __user *p,
                           size_t count, loff_t *pos)
{
    if (file->f_op->write)
        return file->f_op->write(file, p, count, pos);
    else if (file->f_op->write_iter)
        return new_sync_write(file, p, count, pos);
    else
        return -EINVAL;
}
```

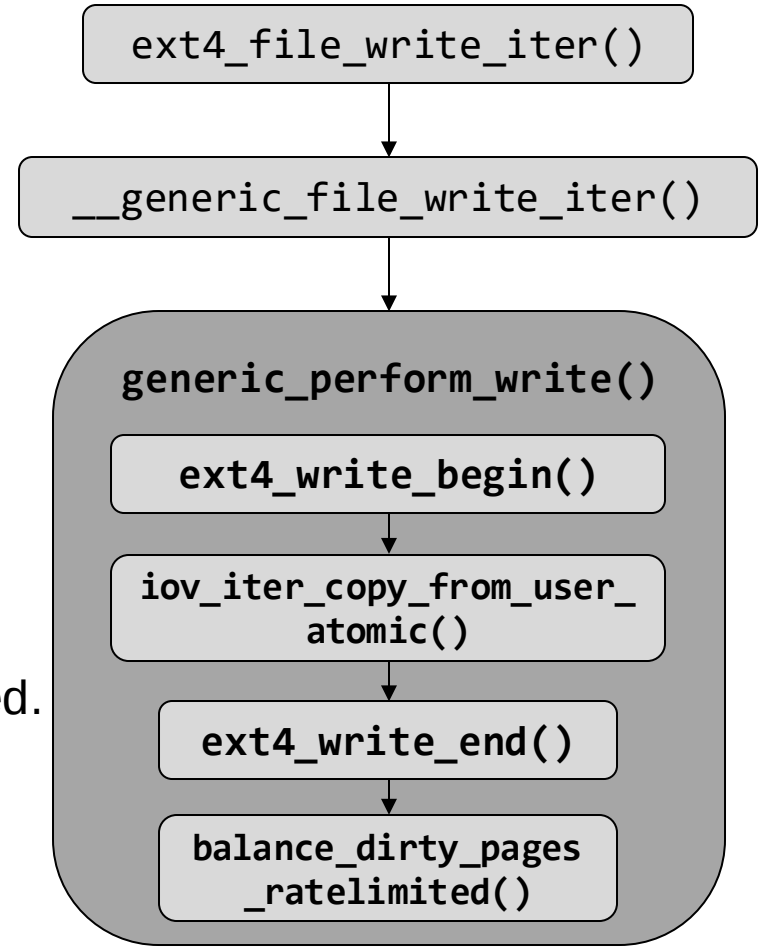
- **__vfs_write()**
  - Select actual write function in the filesystem with `struct file_operation` of file
  - `write()` or `write_iter()` function pointer will be called

Connecting between VFS and Ext4



General write routine for one iteration

- **generic_perform_write()**
  1. **write_begin**
     - Get page from page cache
  2. **iov_iter_copy_from_user_atomic**
     - Write user space data into page
  3. **write_end**
     - Mark page as dirty
  4. **balance_dirty_pages_ratelimited**
     - Periodically check the system's dirty state and will initiate writeback if needed.
     - Processes which are dirtying memory should call in here once for each page which was newly dirtied



When do we balance dirty pages?

- **sysctl**
  - Configure kernel parameters at runtime

```
$ sysctl vm
```

  - **`vm.dirty_background_ratio`**
    - Start background writeback (via writeback threads) at this percentage
    - Default: 10
  - **`vm.dirty_ratio`**
    - The generator of dirty data starts writeback at this percentage
    - Default: 20

vm.dirty_ratio Test

- **Extract generic_perform_write() into pxt4 module**
  1. `mkdir <pxt4 dir>/mm`
  2. `cd <linux src dir>/mm`
  3. `cp filemap.c internal.h <pxt4 dir>/mm`
  4. `cd <pxt4 dir>`
  5. In <pxt4 dir>/mm/filemap.c, delete every function except:
     - `generic_perform_write()`
     - `__generic_file_write_iter()`
  6. Change function names into `pxt4_*`
  7. Remove `EXPORT_SYMBOL` macros
  8. Add calclock to measure `balance_dirty_pages_ratelimited()` function time
  9. Call our functions from `pxt4_file_write_iter()`
  10. Add mm/filemap.o into Makefile and make pxt4 module

vm.dirty_ratio Test

❖ <pxt4 dir>/mm/filemap.c

```
...
//#define CREATE_TRACE_POINTS
//#include <trace/events/filemap.h>
...
#include <asm/mman.h>

#if 0
static void page_cache_delete(struct address_space *mapping,
...
EXPORT_SYMBOL(grab_cache_page_write_begin);
#endif

ssize_t pxt4_generic_perform_write(struct file *file,
                struct iov_iter *i, loff_t pos)
{
    ...
}
//EXPORT_SYMBOL(generic_perform_write);

ssize_t __pxt4_generic_file_write_iter(struct kiocb *iocb, struct iov_iter *from)
{
   ...
}
//EXPORT_SYMBOL(__generic_file_write_iter);

#if 0
ssize_t generic_file_write_iter(struct kiocb *iocb, struct iov_iter *from)
...
EXPORT_SYMBOL(try_to_release_page);
#endif
```

vm.dirty_ratio Test

- <pxt4 dir>/mm/filemap.c

```
ssize_t pxt4_generic_perform_write(struct file *file,
                struct iov_iter *i, loff_t pos)
{
    ...
    do {
        ...
        balance_dirty_pages_ratelimited(mapping);
    } while (iov_iter_count(i));

    return written ? written : status;
}

ssize_t __pxt4_generic_file_write_iter(struct kiocb *iocb, struct iov_iter *from)
{
        ...
        if (iocb->ki_flags & IOCB_DIRECT) {
                ...
        } else {
                written = pxt4_generic_perform_write(file, from, iocb->ki_pos);
                if (likely(written > 0))
                        iocb->ki_pos += written;
        }
        ...
}
```

Add calclock here

vm.dirty_ratio Test

- <pxt4 dir>/file.c

```
...
ssize_t __pxt4_generic_file_write_iter(struct kiocb *iocb, struct iov_iter *from);

static ssize_t
pxt4_file_write_iter(struct kiocb *iocb, struct iov_iter *from)
{
    ...
    ret = __pxt4_generic_file_write_iter(iocb, from);
    ...
}
...
```

vm.dirty_ratio Test

- **Run Fio test with vm.dirty_ratio = 20 (default)**
  1. sudo insmod <jbd3 dir>/jbd3.ko
  2. sudo insmod <pxt4 dir>/pxt4.ko
  3. sudo mount –t pxt4 <testing device> <mount point>
  4. sudo fio seq-write.fio
  5. sudo umount <mount point>
  6. sudo rmmod pxt4
- **Run test with vm.dirty_ratio = 40**

```
$ sudo sysctl vm.dirty_ratio=40
```

  - **1~6** above
- **Run test with vm.dirty_ratio = 100**

```
$ sudo sysctl vm.dirty_ratio=100
```

  - **1~6** above

vm.dirty_ratio Test

Results

- vm.dirty_ratio = **20**

```
file_write_iter is called 3,145,728 times, and the time interval is 148,506,252,395ns
balance_dirty_pages_ratelimited is called 3,145,728 times, and the time interval is 13,992,713,303ns
```

  - Total time: 148 / (4 thr) = 37s
  - BDPR time: 13 / (4 thr) = 3.3s

- vm.dirty_ratio = **40**

```
file_write_iter is called 3,145,728 times, and the time interval is 133,019,911,099ns
balance_dirty_pages_ratelimited is called 3,145,728 times, and the time interval is 8,285,047,904ns
```

  - Total time: 133 / (4 thr) = 33s
  - BDPR time: 8 / (4 thr) = 2s

- vm.dirty_ratio = **100**

```
file_write_iter is called 3,145,728 times, and the time interval is 128,953,475,869ns
balance_dirty_pages_ratelimited is called 3,145,728 times, and the time interval is 12,204,476,217ns
```

  - Total time: 129 / (4 thr) = 32s
  - BDPR time: 12 / (4 thr) = 3s